

Proceedings of the  
**14th International Workshop on Confluence**

September 2–3, 2025

Leipzig, Germany

## Foreword

The 14th International Workshop on Confluence (IWC 2025) is held on September 2–3, 2025, in Leipzig, Germany, co-located with the 20th International Workshop on Termination (WST 2025).

Confluence, as a general notion of determinism, is an essential property of rewrite systems and has emerged as a crucial concept for many applications. However, the confluence property is also relevant to various further areas of rewriting, such as completion, commutation, termination, modularity, and complexity. The International Workshop on Confluence was created as a forum to discuss all these aspects, as well as related topics, implementation issues, and new applications.

IWC 2025 continues this tradition. The present report comprises seven regular submissions, the abstracts of two invited talks by Salvador Lucas and Vincent van Oostrom, and the abstract of an IWC/WST joint invited talk by Aart Middeldorp, as well as descriptions of tools participating in the 14th Confluence Competition (CoCo 2025). The contributions in these proceedings reflect the wide scope of current research on confluence, ranging from new confluence criteria and novel confluence-related properties over formalization of confluence results to implementation aspects and applications. At the same time, the spectrum of rewrite formalisms (first- as well as higher-order, conditional rewriting, rewriting under strategies) used to model problems from different application areas underlines the importance of confluence for various domains.

The renewed interest in confluence research in the last decade resulted in a variety of novel approaches, which were also implemented in powerful tools that compete in the annual confluence competition. The second part of this report devoted to CoCo 2025 provides a general overview as well as system descriptions of all competition entrants.

IWC 2025 was made possible by the commitment of many people who contributed to the submissions, the preparation and the program of the workshop, as well as the confluence competition. These include authors of papers and tools, committee members, external reviewers, and the organizers of CoCo, as well as the local organizers. Their hard work is very much appreciated.

Raúl Gutiérrez and Naoki Nishida

Valencia and Nagoya, 31 August 2025

## Organization

### IWC Steering Committee

- Mauricio Ayala-Rincón, Brasilia University, Brasil
- Sarah Winkler, Free University of Bozen-Bolzano, Italy

### IWC Program Committee

- Takahito Aoto, Niigata University, Japan
- Thiago Felicissimo, INRIA, France
- Carsten Fuhs, Birkbeck, University of London, UK
- Raúl Gutiérrez, Universitat Politècnica de València, Spain (co-chair)
- Maja Hanne Kirkeby, Roskilde University, Denmark
- Sandra Alves, Universidade do Porto, Portugal
- Naoki Nishida, Nagoya University, Japan (co-chair)
- René Thiemann, University of Innsbruck, Austria
- Femke van Raamsdonk, Vrije Universiteit Amsterdam, Netherlands

### CoCo Steering Committee

- Raúl Gutiérrez, Universitat Politècnica de València, Spain
- Aart Middeldorp, University of Innsbruck, Austria
- Naoki Nishida, Nagoya University, Japan
- Teppei Saito, JAIST, Japan
- René Thiemann, University of Innsbruck, Austria

# Contents

<b>Foreword</b>	<b>ii</b>
<b>Organization</b>	<b>iii</b>
<b>Abstracts of Invited Talks</b>	<b>1</b>
Confluence of Conditional Rewriting Modulo	
<i>Salvador Lucas</i> . . . . .	1
Conway’s Game of Life and other orthogonal rewrite systems	
<i>Vincent van Oostrom</i> . . . . .	2
<b>Abstract of IWC/WST Joint Invited Talk</b>	<b>4</b>
Termination and Confluence: Remembering Hans Zantema	
<i>Aart Middeldorp</i> . . . . .	4
<b>Workshop Contributions</b>	<b>5</b>
Improving Confluence Analysis for LCTRSs	
<i>Jonas Schöpfung and Aart Middeldorp</i> . . . . .	5
Towards Confluence of Deterministic Higher-Order Pattern Rewrite Systems by Critical Pairs	
<i>Johannes Niederhauser and Aart Middeldorp</i> . . . . .	12
Term Evaluation Systems with Refinements for Contextual Improvement by Critical Pair Analysis	
<i>Makoto Hamana and Koko Muroya</i> . . . . .	19
Confluence of 001- and 101-infinitary $\lambda$ -calculi by linear approximation	
<i>Rémy Cerda and Lionel Vaux Auclair</i> . . . . .	24
Deciding pattern completeness in non-deterministic polynomial time	
<i>René Thiemann and Akihisa Yamada</i> . . . . .	31
Proving and disproving feasibility with infChecker	
<i>Raúl Gutiérrez and Salvador Lucas</i> . . . . .	38
Redeeming Newman; orthogonality in rewriting; Past, present and future in a 1-algebraic setting	
<i>Vincent van Oostrom</i> . . . . .	45
<b>Confluence Competition</b>	<b>52</b>
Confluence Competition 2025	
<i>Raúl Gutiérrez, Aart Middeldorp, Naoki Nishida, Teppei Saito, and René Thiemann</i> . . . . .	52
CSI-Grackle	
<i>Liao Zhang and Qinxiang Cao</i> . . . . .	54
crest 1.0	
<i>Jonas Schöpfung and Aart Middeldorp</i> . . . . .	55
Natto: a small infeasibility prover based on term orders	
<i>Teppei Saito</i> . . . . .	56
AProVE25: Confluence Analysis in a Termination Tool	
<i>Jan-Christoph Kassing and Tobias Sokolowski</i> . . . . .	57
Hakusan 0.12: A Confluence Tool	
<i>Fuyuki Kawano, Hiroka Hondo, Nao Hirokawa, and Kiraku Shintani</i> . . . . .	58
CONFident at the 2025 Confluence Competition	
<i>Raúl Gutiérrez and Salvador Lucas</i> . . . . .	60
infChecker at the 2025 Confluence Competition	
<i>Raúl Gutiérrez and Salvador Lucas</i> . . . . .	61
CeTA 3.6	
<i>Christina Kirk and René Thiemann</i> . . . . .	62
ACP: System Description for CoCo 2025	
<i>Takahito Aoto</i> . . . . .	64
CO3 (Version 2.6)	
<i>Naoki Nishida and Misaki Kojima</i> . . . . .	65

CRaris (Version 1.1)	
<i>Naoki Nishida and Misaki Kojima</i>	67
FORT-h 2.1	
<i>Fabian Mitterwallner and Aart Middeldorp</i>	69
CSI 1.2.7	
<i>Fabian Mitterwallner and Aart Middeldorp</i>	70
FORTify 2.1	
<i>Fabian Mitterwallner and Aart Middeldorp</i>	71
The System SOL	
<i>Makoto Hamana and Shotaro Karasaki</i>	72

# Confluence of Conditional Rewriting Modulo\*

Salvador Lucas

DSIC & VRAIN, Universitat Politècnica de València, Spain  
slucas@dsic.upv.es

## Abstract

Sets of equations  $E$  play an important computational role in rewriting-based systems  $\mathcal{R}$  by defining an equivalence relation  $=_E$  inducing a partition of terms into  $E$ -equivalence classes on which rewriting computations, denoted  $\rightarrow_{\mathcal{R}/E}$  and called *rewriting modulo  $E$*  [1, 8, 3, 7, 4], are issued. This paper investigates *confluence of  $\rightarrow_{\mathcal{R}/E}$* , usually called  *$E$ -confluence*, for *conditional* rewriting-based systems, where rewriting steps are determined by conditional rules. We rely on Jouannaud and Kirchner’s framework [5] to investigate confluence of an abstract relation  $R$  modulo an abstract equivalence relation  $E$  on a set  $A$ . We show how to particularize the framework to be used with conditional systems. Then, we show how to define appropriate finite sets of *conditional pairs* to prove and disprove  $E$ -confluence [6]. Our results apply to well-known classes of rewriting-based systems. In particular, to *Equational (Conditional) Term Rewriting Systems* [2].

## References

- [1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. Code Optimization And Finite Church-Rosser Theorems. In Randall Rustin, editor, *Design and Oprimization of Compilers, New York, March 29-30 1971*, volume 5 of *Courant Computer Science Symposium*, pages 89–105. Prentice-Hall, 1972.
- [2] Francisco Durán and José Meseguer. On the church-rosser and coherence properties of conditional order-sorted rewrite theories. *J. Log. Algebraic Methods Program.*, 81(7-8):816–850, 2012.
- [3] Gérard P. Huet. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *J. ACM*, 27(4):797–821, 1980.
- [4] Jean-Pierre Jouannaud. Confluent and coherent equational term rewriting systems: Application to proofs in abstract data types. In Giorgio Ausiello and Marco Protasi, editors, *CAAP’83, Trees in Algebra and Programming, 8th Colloquium, Proceedings*, volume 159 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 1983.
- [5] Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM J. Comput.*, 15(4):1155–1194, 1986.
- [6] Salvador Lucas. Confluence of Conditional Rewriting Modulo. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:21, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [7] Gerald E. Peterson and Mark E. Stickel. Complete sets of reductions for some equational theories. *J. ACM*, 28(2):233–264, 1981.
- [8] Ravi Sethi. Testing for the Church-Rosser Property. *J. ACM*, 21(4):671–679, 1974.

---

\*Partially supported by MCIN/AEI project PID2021-122830OB-C42 funded by MCIN/AEI/10.13039/501100011033 and by “ERDF A way of making Europe” and by the grant CIPROM/2022/6 funded by Generalitat Valenciana.

# Conway's Game of Life and other orthogonal rewrite systems

Vincent van Oostrom

University of Sussex, UK  
Vincent.van-Oostrom@sussex.ac.uk

We show how Conway's Game of Life (GoL) can be modelled by means of orthogonal graph rewriting. More precisely, we model GoL by means of a graph rewrite system where each cell of the grid is represented by a node having 8 ports, each linked to one of its 8 neighbouring nodes. A rewrite rule then lets a node cyclically rotate its principal port to the next (either widdershins or deosil) port while updating its *alive-neighbour-count*. After a full rotation, its state is updated accordingly. We show this yields a graph rewrite system (GRS) where computing the next GoL-iteration may be achieved in 8 ticks by means of what we call *@locksteps* better known in rewriting as *full multisteps* contracting *all* redex-patterns in the graph in parallel. The GRS being *orthogonal* liberates one from *having to* perform *@locksteps* only, to always contract *all* redex-patterns: orthogonality makes that redex-patterns may be contracted asynchronously, even one at the time, need not be contracted in *lockstep*. There are no *clogsteps* (so to speak), making the modelling ideally suited for implementation on GPUs.

In the second part of the presentation we show in what sense the graph rewrite system used to model GoL in the first part is *orthogonal*. We show it constitutes a so-called Interaction Net (IN) and that a (single) step from graph  $G$  to graph  $H$  with respect to rule  $\rho : L \rightarrow R$  can be decomposed into three phases:

1. the *matching* phase, an SC-expansion  $G_{SC} \leftarrow C[L]$  exhibiting the to-be-replaced substructure  $L$  within  $G$ ;
2. the *replacement*  $C[L] \rightarrow C[R]$  of the exhibited substructure  $L$ , left-hand side of rule  $\rho$ , by its right-hand side  $R$ ;
3. the *substitution* phase, an SC-reduction  $C[R] \rightarrow_{SC} H$  plugging-in the substructure  $R$  yielding  $H$ .

Here SC stands for *Substitution Calculus*, the calculus used for abstracting subgraphs into variables and substituting for them again (matching and substitution). In the case of INs the SC is particularly simple, and consists essentially in managing *indirection* nodes. We exemplify this decomposition extends to term rewrite systems (TRSs; having the simply typed  $\lambda\alpha\beta\eta$ -calculus as SC) and to term graph rewrite systems (TGRSs; having an SC, the  $\mathcal{K}$ -calculus, based on sharing), putting INs on a par with orthogonal TRSs and orthogonal TGRSs, thereby unlocking the theory of orthogonality to GoL and other cellular automata. For instance, that INs are confluent, even has least upperbounds, is an immediate consequence of orthogonality (confluence-by-parallelism).

**References:** there being an abundance of literature on Game of Life and on Interaction Nets, we only give references to the lesser-known approach to structured rewriting by means of Substitution Calculi: [2], [4], [1], and [3].

## References

- [1] Nao Hirokawa, Julian Nagele, Vincent van Oostrom, and Michio Oyamaguchi. Confluence by critical pair analysis revisited. In Pascal Fontaine, editor, *Proceedings of the 27th International Conference on Automated Deduction*, volume 11716 of *Lecture Notes in Computer Science*, pages 319–336. Springer, 2019. doi:10.1007/978-3-030-29436-6\_19.
- [2] Vincent val Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. Phd-thesis – research and graduation internal, Vrije Universiteit Amsterdam, 1994. URL: <https://research.vu.nl/en/publications/confluence-for-abstract-and-higher-order-rewriting>.
- [3] Vincent van Oostrom. Accounting for the cost of substitution in structured rewriting, 2025. Invited talk at the 12th International Workshop on Higher-Order Rewriting. URL: <http://www.javakade.nl/research/talk/horpresentation1407025.pdf>.
- [4] Femke van Raamsdonk. *Confluence and Normalisation of Higher-Order Rewriting*. Phd-thesis – research and graduation internal, Vrije Universiteit Amsterdam, 1996. URL: <https://research.vu.nl/en/publications/confluence-and-normalisation-of-higher-order-rewriting>.



# Termination and Confluence: Remembering Hans Zantema

Aart Middeldorp

University of Innsbruck, Austria  
`aart.middeldorp@uibk.ac.at`

In this presentation I give an incomplete overview of the many contributions of Hans Zantema<sup>1</sup> to termination and confluence. Several of these were presented at earlier workshops on termination<sup>2</sup> and confluence,<sup>3</sup> and I include a biased overview of the development of these workshops and associated competitions [1,2].

## References

- [1] Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The Termination and Complexity Competition. *Proc. 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 11429 of *LNCS*, pages 156–166, 2019. doi: [10.1007/978-3-030-17502-3\\_10](https://doi.org/10.1007/978-3-030-17502-3_10).
- [2] Aart Middeldorp, Julian Nagele, and Kiraku Shintani. CoCo 2019: Report on the Eighth Confluence Competition. *International Journal on Software Tools for Technology Transfer*, 2021. doi: [10.1007/s10009-021-00620-4](https://doi.org/10.1007/s10009-021-00620-4).

---

<sup>1</sup><https://hzantema.win.tue.nl/>

<sup>2</sup><https://termination-portal.org/wiki/WST>

<sup>3</sup><http://cl-informatik.uibk.ac.at/iwc/>

# Improving Confluence Analysis for LCTRSs\*

Jonas Schöpfung and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Innsbruck, Tirol, Austria  
{jonas.schoepf,aart.middeldorp}@uibk.ac.at

## Abstract

We lift two well-known confluence techniques—the redundant rules technique and the reduction method—from term rewrite systems to logically constrained term rewrite systems. We establish sufficient criteria for both techniques in the constrained setting, increasing flexibility of confluence analysis.

## 1 Introduction

Confluence is a fundamental property of rewrite systems that ensures unique results of rewriting, independent of the order in which rewrite rules are applied. Proving confluence is thus central to ensure unique computations and has applications in areas such as programming language semantics, formal verification, and theorem proving. While confluence of term rewrite systems (TRSs) has been extensively studied, for logically constrained rewrite systems (LCTRSs)—an extension that incorporates logical constraints into rules—only in recent years there was notable progress. This includes several well-known critical pair criteria like strong closedness [5] or development closedness [7]. Recently, a criterion to show non-confluence of LCTRSs, based on finding a non-joinable constrained critical pair, was introduced in [6]. A key challenge in the confluence analysis of LCTRSs lies in efficient automation of the aforementioned methods. This is especially difficult because existing techniques lack support for modular reasoning. However, precisely those methods that provide modularity for confluence analysis of TRSs have shown to significantly improve automation.

In this paper we extend two transformation techniques for TRSs, redundant rules [4] and the reduction method [8], to LCTRSs. We expect that automating these will improve the performance of LCTRS confluence tools.

## 2 Preliminaries

Due to space constraints, we assume familiarity with term rewriting and recall only key notions for LCTRSs. For background on TRSs and LCTRSs, see [1] and [7], respectively. We assume a many-sorted signature  $\mathcal{F} = \mathcal{F}_{\text{te}} \cup \mathcal{F}_{\text{th}}$  and a non-empty set of values  $\mathcal{Val}$  where  $\mathcal{F}_{\text{te}} \cap \mathcal{F}_{\text{th}} \subseteq \emptyset$  and  $\mathcal{Val} \subseteq \mathcal{F}_{\text{th}}$ . A *constrained rewrite rule* is a triple  $\ell \rightarrow r [\varphi]$  where  $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$  are terms of the same sort such that  $\text{root}(\ell) \in \mathcal{F}_{\text{te}} \setminus \mathcal{F}_{\text{th}}$  and  $\varphi$  is a constraint. We denote the set  $\mathcal{Var}(\varphi) \cup (\mathcal{Var}(r) \setminus \mathcal{Var}(\ell))$  of *logical* variables in  $\ell \rightarrow r [\varphi]$  by  $\mathcal{LVar}(\ell \rightarrow r [\varphi])$ . A constrained rewrite rule is left-linear if non-logical variables in the left-hand side occur at most once. A substitution  $\sigma$  is said to *respect* a rule  $\ell \rightarrow r [\varphi]$ , denoted by  $\sigma \models \ell \rightarrow r [\varphi]$ , if  $\text{Dom}(\sigma) \subseteq \mathcal{Var}(\ell) \cup \mathcal{Var}(r) \cup \mathcal{Var}(\varphi)$ ,  $\sigma(x) \in \mathcal{Val}$  for all  $x \in \mathcal{LVar}(\ell \rightarrow r [\varphi])$ , and  $\llbracket \varphi \sigma \rrbracket = \top$ . Moreover, a constraint  $\varphi$  is respected by  $\sigma$ , denoted by  $\sigma \models \varphi$ , if  $\sigma(x) \in \mathcal{Val}$  for all  $x \in \mathcal{Var}(\varphi)$  and  $\llbracket \varphi \sigma \rrbracket = \top$ . We call  $f(x_1, \dots, x_n) \rightarrow y [y = f(x_1, \dots, x_n)]$  with a fresh variable  $y$  and

---

\*This research is funded by the Austrian Science Fund (FWF) project I 5943-N.

$f \in \mathcal{F}_{\text{th}} \setminus \text{Val}$  a *calculation rule*. The set of calculation rules is denoted by  $\mathcal{R}_{\text{ca}}$ . Those rules are not included in  $\mathcal{R}$ , but we define  $\mathcal{R}_{\text{rc}}$  as the union of  $\mathcal{R}$  and  $\mathcal{R}_{\text{ca}}$ . A rewrite step  $s \rightarrow_{\mathcal{R}} t$  satisfies  $s|_p = \ell\sigma$  and  $t = s[r\sigma]_p$  for some position  $p$ , constrained rewrite rule  $\ell \rightarrow r [\varphi]$  in  $\mathcal{R}_{\text{rc}}$ , and substitution  $\sigma$  such that  $\sigma \models \ell \rightarrow r [\varphi]$ .

A *constrained term* is a pair  $s [\varphi]$  consisting of a term  $s$  and a constraint  $\varphi$ . Two constrained terms  $s [\varphi]$  and  $t [\psi]$  are *equivalent*, denoted by  $s [\varphi] \sim t [\psi]$ , if for every substitution  $\gamma \models \varphi$  with  $\text{Dom}(\gamma) = \text{Var}(\varphi)$  there is some substitution  $\delta \models \psi$  with  $\text{Dom}(\delta) = \text{Var}(\psi)$  such that  $s\gamma = t\delta$ , and vice versa. If  $s|_p = \ell\sigma$  for some constrained rewrite rule  $\rho: \ell \rightarrow r [\psi] \in \mathcal{R}_{\text{rc}}$ , position  $p$ , and substitution  $\sigma$  such that  $\sigma(x) \in \text{Val} \cup \text{Var}(\varphi)$  for all  $x \in \mathcal{LVar}(\rho)$ ,  $\varphi$  is satisfiable and  $\varphi \Rightarrow \psi\sigma$  is valid then  $s [\varphi] \rightarrow_{\mathcal{R}} s[r\sigma]_p [\varphi]$ . The rewrite relation  $\rightarrow_{\mathcal{R}}$  on constrained terms is defined as  $\sim \cdot \rightarrow_{\mathcal{R}} \cdot \sim$ . We write  $s [\varphi] \rightarrow_{\mathcal{R}}^* t [\psi]$  if the position in the rewrite step is below position  $p$ .

Given a constrained rewrite rule  $\rho: \ell \rightarrow r [\varphi]$ , we write  $\mathcal{EC}_{\rho}$  for  $\bigwedge \{x = x \mid x \in \mathcal{EVar}(\rho)\}$ . Here  $\mathcal{EVar}(\rho)$  denotes the set  $\text{Var}(r) \setminus (\text{Var}(\ell) \cup \text{Var}(\varphi))$  of extra variables of  $\rho$ . An *overlap* of an LCTRS  $\mathcal{R}$  is a triple  $\langle \rho_1, p, \rho_2 \rangle$  with rules  $\rho_1: \ell_1 \rightarrow r_1 [\varphi_1]$  and  $\rho_2: \ell_2 \rightarrow r_2 [\varphi_2]$ , satisfying the following conditions:

- (1)  $\rho_1$  and  $\rho_2$  are variable-disjoint variants of rewrite rules in  $\mathcal{R}_{\text{rc}}$ ,
- (2)  $p \in \text{Pos}_{\mathcal{F}}(\ell_2)$ ,
- (3)  $\ell_1$  and  $\ell_2|_p$  unify with mgu  $\sigma$  such that  $\sigma(x) \in \text{Val} \cup \mathcal{V}$  for all  $x \in \mathcal{LVar}(\rho_1) \cup \mathcal{LVar}(\rho_2)$ ,
- (4)  $\varphi_1\sigma \wedge \varphi_2\sigma$  is satisfiable, and
- (5) if  $p = \epsilon$  then  $\rho_1$  and  $\rho_2$  are not variants, or  $\text{Var}(r_1) \not\subseteq \text{Var}(\ell_1)$ .

In this case we call

$$\ell_2\sigma[r_1\sigma]_p \approx r_2\sigma [\varphi_1\sigma \wedge \varphi_2\sigma \wedge \psi\sigma]$$

a *constrained critical pair* obtained from the overlap  $\langle \rho_1, p, \rho_2 \rangle$ . Here  $\psi = \mathcal{EC}_{\rho_1} \wedge \mathcal{EC}_{\rho_2}$ . The set of all constrained critical pairs of  $\mathcal{R}$  is denoted by  $\text{CCP}(\mathcal{R})$ . A constrained critical pair  $s \approx t [\varphi]$  is *trivial* if  $s\sigma = t\sigma$  for every substitution  $\sigma$  with  $\sigma \models \varphi$ .

We conclude this section by recalling a transformation from [7, Definition 1] which transforms an LCTRS  $\mathcal{R}$  to a (possibly infinite) TRS  $\overline{\mathcal{R}}$ . Note that after applying this transformation we obtain that  $\rightarrow_{\mathcal{R}}$  and  $\rightarrow_{\overline{\mathcal{R}}}$  coincide for an LCTRS  $\mathcal{R}$  ([7, Lemma 1]).

**Definition 1.** Given an LCTRS  $\mathcal{R}$ , the TRS  $\overline{\mathcal{R}}$  consists of the following rules:  $\ell\tau \rightarrow r\tau$  for all  $\rho: \ell \rightarrow r [\varphi] \in \mathcal{R}_{\text{rc}}$  with  $\tau \models \rho$  and  $\text{Dom}(\tau) = \mathcal{LVar}(\rho)$ .

### 3 Redundant Constrained Rules

In this section we lift the redundant rules technique [4] to the constrained setting. The idea is to remove rules that hinder the confluence proof or to add rules that make other confluence methods applicable. Two LCTRSs  $\mathcal{R}$  and  $\mathcal{R}'$  are said to share the same theory if they differ only in  $\mathcal{F}_{\text{te}}$  and their respective rule sets  $\mathcal{R}$  and  $\mathcal{R}'$ . We denote this by  $\mathcal{R} \simeq \mathcal{R}'$ . For the remainder of this section, we assume  $\mathcal{R} \simeq \mathcal{S}$  for the LCTRSs  $\mathcal{R}$  and  $\mathcal{S}$ .

**Definition 2.** A rule  $\rho: \ell \rightarrow r [\varphi] \in \mathcal{R}$  is *redundant* if

$$\ell \approx r [\varphi \wedge \mathcal{EC}_{\rho}] \rightarrow_{\mathcal{R} \setminus \{\rho\}, \geq 1}^* \ell' \approx r' [\psi]$$

for some trivial  $\ell' \approx r' [\psi]$ . A set of constrained rules  $\mathcal{S}$  is redundant in  $\mathcal{R}$  if all rules  $\ell \rightarrow r [\varphi] \in \mathcal{S}$  are redundant in  $\mathcal{R}$ .

**Example 3.** Consider an LCTRS  $\mathcal{R}$  over the theory of integers and the constrained rewrite rule  $\rho: f(x+x) \rightarrow f(z) [z = 2 \cdot x] \in \mathcal{R}$ . This rule is redundant as witnessed by the following rewrite sequence using the calculation rule  $x' + y' \rightarrow z' [z' = x' + y']$ :

$$\begin{aligned} f(x+x) &\approx f(z) [z = 2 \cdot x] \sim f(x+x) \approx f(z) [z = 2 \cdot x \wedge z' = x+x] \\ &\rightarrow_{\mathcal{R} \setminus \{\rho\}, \geq 1} f(z') \approx f(z) [z = 2 \cdot x \wedge z' = x+x] \end{aligned}$$

The resulting constrained equation is clearly trivial and hence Definition 2 is satisfied.

A perhaps non-obvious fact is that the removal of redundant rules can actually transform a non-left-linear LCTRS into a left-linear one. Since most confluence criteria require left-linearity, they may then become applicable.

**Theorem 4.** *If a set of constrained rules  $\mathcal{S}$  is redundant in an LCTRS  $\mathcal{R}$  then  $\mathcal{R}$  is confluent if and only if  $\mathcal{R} \cup \mathcal{S}$  is confluent.*

*Proof.* We first show  $\rightarrow_{\mathcal{R}}^* = \rightarrow_{\mathcal{R} \cup \mathcal{S}}^*$ . The inclusion  $\rightarrow_{\mathcal{R}}^* \subseteq \rightarrow_{\mathcal{R} \cup \mathcal{S}}^*$  is trivial. For the reverse inclusion, it suffices to show  $\rightarrow_{\mathcal{S}} \subseteq \rightarrow_{\mathcal{R}}^*$ . Consider a term  $s$ , a constrained rule  $\rho: \ell \rightarrow r [\varphi] \in \mathcal{S}$ , a position  $p$  and a substitution  $\sigma \models \rho$  such that  $s = s[\ell\sigma]_p \rightarrow_{\mathcal{S}} s[r\sigma]_p$ . By redundancy of  $\rho$  we obtain  $\ell \approx r [\varphi] \rightsquigarrow_{\mathcal{R} \setminus \{\rho\}, \geq 1}^* \ell' \approx r' [\psi]$  for a trivial  $\ell' \approx r' [\psi]$ . Repeated application of [7, Lemma 2] gives  $\ell\sigma \rightarrow_{\mathcal{R} \setminus \{\rho\}}^* \ell'\gamma$  and  $r\sigma = r'\gamma$  for a substitution  $\gamma \models \psi$ . Moreover, by triviality [5, Definition 5] we obtain  $\ell'\gamma = r'\gamma$  and thus  $\ell\sigma \rightarrow_{\mathcal{R} \setminus \{\rho\}}^* \ell'\gamma = r'\gamma = r\sigma$ . Closure under contexts yields  $s = s[\ell\sigma]_p \rightarrow_{\mathcal{R} \setminus \{\rho\}}^* s[r\sigma]_p$ . Hence  $\rightarrow_{\mathcal{S}} \subseteq \rightarrow_{\mathcal{R}}^*$  as desired.  $\square$

From the proof of this theorem it follows that  $\rightarrow_{\mathcal{R}} = \rightarrow_{\mathcal{R} \setminus \{\rho\}}$  in Example 3. By changing reduction to conversion in the definition of redundant rules, we obtain the following variant.

**Theorem 5.** *If  $\mathcal{R}$  is confluent,  $\ell \approx r [\varphi \wedge \mathcal{EC}_\rho] \leftrightarrow_{\mathcal{R} \setminus \{\rho\}, > \epsilon}^* \ell' \approx r' [\psi]$  and  $\ell' \approx r' [\psi]$  is trivial for every rule  $\rho: \ell \rightarrow r [\varphi] \in \mathcal{S}$  then  $\mathcal{R} \cup \mathcal{S}$  is confluent.*

*Proof.* We first show  $\rightarrow_{\mathcal{S}} \subseteq \leftrightarrow_{\mathcal{R}}^*$ . Let  $s = s[\ell\sigma]_p \rightarrow_{\mathcal{S}} s[r\sigma]_p$  for some rule  $\rho: \ell \rightarrow r [\varphi] \in \mathcal{S}$ , position  $p$  and substitution  $\sigma \models \rho$ . The assumption on  $\rho$  yields  $\ell \approx r [\varphi] \leftrightarrow_{\mathcal{R}', > \epsilon}^* \ell' \approx r' [\psi]$  for some trivial  $\ell' \approx r' [\psi]$  and  $\mathcal{R}' = \mathcal{R} \setminus \{\rho\}$ . Repeated application of [9, Lemma 6] (and its symmetric version) yields  $\ell\sigma \leftrightarrow_{\mathcal{R}'}^* \ell'\gamma$  and  $r\sigma \leftrightarrow_{\mathcal{R}'}^* r'\gamma$  for some  $\gamma \models \psi$ . Triviality of  $\ell' \approx r' [\psi]$  gives  $\ell'\gamma = r'\gamma$  and thus  $\ell\sigma \leftrightarrow_{\mathcal{R}'}^* \ell'\gamma = r'\gamma \leftrightarrow_{\mathcal{R}'}^* r\sigma$ . Hence  $s = s[\ell\sigma]_p \leftrightarrow_{\mathcal{R}'}^* s[r\sigma]_p$  by closure under contexts. From  $\rightarrow_{\mathcal{S}} \subseteq \leftrightarrow_{\mathcal{R}}^*$  we obtain  $\leftrightarrow_{\mathcal{R} \cup \mathcal{S}}^* = \leftrightarrow_{\mathcal{R}}^*$  and thus  $\leftrightarrow_{\mathcal{R} \cup \mathcal{S}}^* \subseteq \downarrow_{\mathcal{R}} \subseteq \downarrow_{\mathcal{R} \cup \mathcal{S}}$  by the confluence of  $\mathcal{R}$ . Hence  $\mathcal{R} \cup \mathcal{S}$  is confluent.  $\square$

In the following example we illustrate that this method indeed strengthens confluence analysis for LCTRSs.

**Example 6.** Consider the LCTRS  $\mathcal{R}$  over the theory of integers with the rules

$$\begin{aligned} \alpha: f(x, y) &\rightarrow x + y [x > 0] & \beta: f(x, y) &\rightarrow d(x, y) [x = 2 * y \wedge y > 0] \\ \gamma: f(x, y) &\rightarrow f(y, x) [x \leq 0] & \delta: d(x, y) &\rightarrow y + x \end{aligned}$$

Note that  $\mathcal{R}$  is left-linear, non-terminating and admits the two constrained critical pairs

$$x + y \approx d(x, y) [x > 0 \wedge x = 2 * y \wedge y > 0] \quad d(x, y) \approx x + y [x = 2 * y \wedge y > 0 \wedge x > 0]$$

Those are neither both strongly closed ([5, Definition 6]), both almost development closed ([7, Definition 6]) nor both 1-parallel closed ([7, Definition 11]). Hence none of the known confluence criteria for LCTRSs [3, 5, 7] apply here. Let us denote the constraint  $x = 2 * y \wedge y > 0$  by  $\varphi$ . We obtain the conversion of  $\beta$

$$\begin{aligned} f(x, y) \approx d(x, y) [\varphi] &\stackrel{\leftrightarrow}{\mathcal{R} \setminus \{\beta\}, > \epsilon} x + y \approx d(x, y) [\varphi] \\ &\stackrel{\leftrightarrow}{\mathcal{R} \setminus \{\beta\}, > \epsilon} x + y \approx y + x [\varphi] \\ &\stackrel{\leftrightarrow}{\mathcal{R} \setminus \{\beta\}, > \epsilon} z \approx y + x [\varphi \wedge z = x + y] \\ &\stackrel{\leftrightarrow}{\mathcal{R} \setminus \{\beta\}, > \epsilon} z \approx z' [\varphi \wedge z = x + y \wedge z' = y + x] \end{aligned}$$

for which the last constrained equation is trivial. The LCTRS  $\mathcal{R} \setminus \{\beta\}$  is orthogonal and therefore confluent. By Theorem 5 we conclude the confluence of  $\mathcal{R}$ .

**Implementation** A prototype implementation following the results of this section is available in the tool *crest* [6]. The following heuristics, which are inspired by [4, Section 5], are currently used; here  $\Rightarrow$  denotes the multi-step rewrite relation [7, Definition 5]:

- (1) for every  $s \approx t [\varphi] \in \text{CCP}(\mathcal{R})$  if it satisfies  $s \approx t [\varphi] \Rightarrow_{\geq 1}^2 \cdot \Rightarrow_{\geq 2}^2 u \approx v [\psi]$  for a trivial  $u \approx v [\psi]$  then we add  $\{s \rightarrow u [\varphi], t \rightarrow v [\varphi]\}$  to the rules,
- (2) for every  $\ell \rightarrow r [\varphi] \in \mathcal{R}$  if  $r [\varphi \wedge \mathcal{EC}_\rho] \Rightarrow^2 r' [\psi]$  then we add the new rule  $\ell \rightarrow r' [\varphi]$  to the rules,
- (3) remove the rule  $\rho: \ell \rightarrow r [\varphi]$  from the set of rules if

$$\ell \approx r [\varphi \wedge \mathcal{EC}_\rho] \Rightarrow_{\mathcal{R} \setminus \{\rho\}, \geq 1}^2 \cdot \Rightarrow_{\mathcal{R} \setminus \{\rho\}, \geq 2}^2 u \approx v [\psi]$$

for a trivial  $u \approx v [\psi]$ .

Note that the first two heuristics are based on Theorem 4, while the last is based on Theorem 5. The confluence problem in Example 6 can be solved by (3).

## 4 Reduction Method for LCTRSs

In this section, we consider compositional confluence analysis for LCTRSs, inspired by the work on TRSs by Shintani and Hirokawa [8]. Our main focus is on the reduction method, which, as the name implies, seeks to reduce a TRS  $\mathcal{R}$  to a smaller subsystem  $\mathcal{C}$  while preserving the confluence property.

Let us recall necessary notions and results from [8]. A TRS  $\mathcal{R}$  is convertible by a TRS  $\mathcal{C}$  if  $\mathcal{C} \subseteq \mathcal{R}$  and for all  $s \approx t \in \text{PCP}(\mathcal{R})$  we have  $s \leftrightarrow_{\mathcal{C}}^* t$ . Here  $\text{PCP}(\mathcal{R})$  denotes the set of parallel critical pairs of  $\mathcal{R}$ .

**Theorem 7** (Theorem 5.4 [8]). *A left-linear TRS  $\mathcal{R}$  is confluent if it is convertible by some confluent TRS  $\mathcal{C}$ .*  $\square$

The results in this section rely on parallel critical pairs. The following example shows why normal critical pairs do not suffice in Theorem 7.

**Example 8.** Consider the TRS  $\mathcal{R}'$  of [2, Example 8] consisting of the rewrite rules

$$\rho_1: f(a, a) \rightarrow b \quad \rho_2: a \rightarrow c \quad \rho_3: f(c, x) \rightarrow f(x, x) \quad \rho_4: f(x, c) \rightarrow f(x, x)$$

We have  $\text{CP}(\mathcal{R}') = \{f(a, c) \approx b, f(c, a) \approx b, f(c, c) \approx f(c, c)\} \subseteq \rightarrow_{\mathcal{C}}^*$  for  $\mathcal{C} = \{\rho_1, \rho_3, \rho_4\}$ . The subsystem  $\mathcal{C}$  is weakly orthogonal and thus confluent. However,  $\mathcal{R}'$  is not confluent as witnessed by  $f(c, c) \xrightarrow{\mathcal{R}'}^* f(a, a) \rightarrow_{\mathcal{R}'} b$ . Note that  $f(c, c) \approx b \in \text{PCP}(\mathcal{R}')$ .

An LCTRS  $\mathcal{C}$  is a *subsystem* of an LCTRS  $\mathcal{R}$ , written as  $\mathcal{C} \sqsubseteq \mathcal{R}$ , if  $\mathcal{C} \simeq \mathcal{R}$  and  $\mathcal{C} \subseteq \mathcal{R}$ . Therefore, these LCTRSs share the same theory and thus have identical mappings, interpretation functions, theory symbols, values, and calculation rules.

**Definition 9.** A constrained parallel critical pair  $s \approx t [\varphi]$  is *convertible* by an LCTRS  $\mathcal{C}$  if  $s \approx t [\varphi] \xleftrightarrow{\mathcal{C}, \geq 1}^* s' \approx t' [\psi]$  for some trivial  $s' \approx t' [\psi]$ . An LCTRS  $\mathcal{R}$  is convertible by  $\mathcal{C}$  if  $\mathcal{C} \sqsubseteq \mathcal{R}$  and all constrained parallel critical pairs  $\text{CPCP}(\mathcal{R})$  of  $\mathcal{R}$  are convertible by  $\mathcal{C}$ .

**Example 10.** The constrained critical pairs in Example 6 are the only constrained parallel critical pairs. Both are convertible by  $\mathcal{C} = \{\alpha, \gamma, \delta\}$  in a single step, e.g.,

$$x + y \approx d(x, y) [x > 0 \wedge \varphi] \rightarrow_{\mathcal{C}} x + y \approx y + x [x > 0 \wedge \varphi]$$

**Theorem 11.** A left-linear LCTRS  $\mathcal{R}$  is confluent if it is convertible by a confluent LCTRS  $\mathcal{C}$ .

*Proof.* Consider a left-linear LCTRS  $\mathcal{R}$  that is convertible by a confluent LCTRS  $\mathcal{C}$ . So  $\mathcal{C} \sqsubseteq \mathcal{R}$ . Employing the transformation in Definition 1 we obtain TRSs  $\overline{\mathcal{R}}$  and  $\overline{\mathcal{C}}$  with  $\rightarrow_{\mathcal{R}} = \rightarrow_{\overline{\mathcal{R}}}$  and  $\rightarrow_{\mathcal{C}} = \rightarrow_{\overline{\mathcal{C}}}$ . Hence  $\overline{\mathcal{C}}$  is confluent and  $\overline{\mathcal{R}}$  is left-linear. We obtain  $\overline{\mathcal{C}} \subseteq \overline{\mathcal{R}}$  from  $\mathcal{C} \sqsubseteq \mathcal{R}$ . We show that  $\overline{\mathcal{R}}$  is convertible by  $\overline{\mathcal{C}}$ . For each parallel critical pair  $s \approx t \in \text{PCP}(\overline{\mathcal{R}})$  there exists a constrained parallel critical pair  $s' \approx t' [\varphi'] \in \text{CPCP}(\mathcal{R})$  with a substitution  $\sigma$  such that  $s'\sigma = s$ ,  $t'\sigma = t$  and  $\sigma \models \varphi'$  by [7, Theorem 2]. Convertibility yields  $s' \approx t' [\varphi] \xleftrightarrow{\mathcal{C}, \geq 1}^* u \approx v [\psi]$  for some trivial  $u \approx v [\psi]$ . Repeatedly applying [5, Lemma 2] to the  $\xrightarrow{\mathcal{C}, \geq 1}$  steps (in both directions) yields that for all substitutions  $\sigma \models \varphi$  there exists a substitution  $\gamma \models \psi$  such that  $s'\sigma \xleftrightarrow{\mathcal{C}}^* u\gamma$  and  $t'\sigma = v\gamma$ . Triviality yields  $u\gamma = v\gamma$  and thus  $s = s'\sigma \xleftrightarrow{\mathcal{C}}^* u\gamma = v\gamma = t'\sigma = t$ . According to Theorem 7  $\overline{\mathcal{R}}$  is confluent which implies the confluence of  $\mathcal{R}$ .  $\square$

To state the next result, we recall some notation from [8]. For TRSs  $\mathcal{R}$  and  $\mathcal{C}$  we write  $\mathcal{R}|_{\mathcal{C}}$  for the TRS  $\{\ell \rightarrow r \in \mathcal{R} \mid \mathcal{F}\text{un}(\ell) \subseteq \mathcal{F}\text{un}(\mathcal{C})\}$ . Here  $\mathcal{F}\text{un}(\mathcal{C}) = \{f \in \mathcal{F}\text{un}(\ell) \cup \mathcal{F}\text{un}(r) \mid \ell \rightarrow r \in \mathcal{C}\}$ . Below we lift the following result of [8] to the constrained setting.<sup>1</sup>

**Theorem 12.** If  $\mathcal{R}|_{\mathcal{C}} \subseteq \rightarrow_{\mathcal{C}}^* \subseteq \rightarrow_{\mathcal{R}}^*$  and  $\mathcal{R}$  is confluent then  $\mathcal{C}$  is confluent.  $\square$

**Definition 13.** For a term  $s$  we define the set of term symbols  $\mathcal{F}\text{un}_{\text{te}}(s)$  in  $s$  as  $\mathcal{F}\text{un}(s) \setminus \mathcal{F}_{\text{th}}$ . Given two LCTRSs  $\mathcal{R}$  and  $\mathcal{C}$  we define  $\mathcal{R}|_{\mathcal{C}} = \{\ell \rightarrow r [\varphi] \in \mathcal{R} \mid \mathcal{F}\text{un}_{\text{te}}(\ell) \subseteq \mathcal{F}\text{un}_{\text{te}}(\mathcal{C})\}$ . Here  $\mathcal{F}\text{un}_{\text{te}}(\mathcal{C}) = \bigcup \{\mathcal{F}\text{un}_{\text{te}}(\ell) \cup \mathcal{F}\text{un}_{\text{te}}(r) \mid \ell \rightarrow r [\varphi] \in \mathcal{C}\}$ . We say that  $\mathcal{R}|_{\mathcal{C}}$  is *simulated* by  $\mathcal{C}$  if every  $\rho: \ell \rightarrow r [\varphi] \in \mathcal{R}|_{\mathcal{C}}$  satisfies  $\ell \approx r [\varphi \wedge \mathcal{E}\mathcal{C}_{\rho}] \xrightarrow{\mathcal{C}, \geq 1}^* u \approx v [\psi]$  for some trivial  $u \approx v [\psi]$ .

**Example 14.** Consider the LCTRS  $\mathcal{R}$  over the theory of integers consisting of the single rule  $f(x) \rightarrow a [x = 1]$ . We have  $\overline{\mathcal{R}} = \{f(1) \rightarrow a\}$ ,  $\mathcal{F}\text{un}_{\text{te}}(\mathcal{R}) = \{f, a\}$  and  $\mathcal{F}\text{un}(\overline{\mathcal{R}}) = \{f, a, 1\}$ .

**Lemma 15.** If  $\mathcal{R}|_{\mathcal{C}}$  is simulated by  $\mathcal{C}$  and  $\mathcal{C} \sqsubseteq \mathcal{R}$  then  $\overline{\mathcal{R}}|_{\overline{\mathcal{C}}} \subseteq \rightarrow_{\overline{\mathcal{C}}}^*$ .

*Proof.* Let  $\mathcal{R}$  and  $\mathcal{C}$  be LCTRSs such that  $\mathcal{C} \sqsubseteq \mathcal{R}$  and  $\mathcal{R}|_{\mathcal{C}}$  is simulated by  $\mathcal{C}$ . Consider  $\rho: \ell \rightarrow r \in \overline{\mathcal{R}}|_{\overline{\mathcal{C}}}$ . Clearly  $\rho \in \overline{\mathcal{R}}$  and  $\mathcal{F}\text{un}(\ell) \subseteq \mathcal{F}\text{un}(\overline{\mathcal{C}})$ . By Definition 1 there exist a constrained rewrite rule  $\rho': \ell' \rightarrow r' [\varphi'] \in \mathcal{R}_{\text{rc}}$  and a substitution  $\sigma$  such that  $\ell = \ell'\sigma$ ,  $r = r'\sigma$  and  $\sigma \models \rho'$

<sup>1</sup>The condition  $\rightarrow_{\mathcal{C}}^* \subseteq \rightarrow_{\mathcal{R}}^*$  is missing in [8, Theorem 8.3] but required: <https://www.jaist.ac.jp/~hirokawa/materials/24lmcs-errata.html>.

with  $\text{Dom}(\sigma) = \mathcal{LVar}(\rho')$ . Note that if  $\rho' \in \mathcal{R}_{\text{ca}}$  then trivially  $\mathcal{C}$  simulates  $\rho'$  as  $\mathcal{C} \sqsubseteq \mathcal{R}$ . Hence it remains to show  $\rho' \in \mathcal{R}|_{\mathcal{C}}$  for  $\rho' \in \mathcal{R}$ , which amounts to  $\mathcal{F}_{\text{un}_{\text{te}}}(\ell') \subseteq \mathcal{F}_{\text{un}_{\text{te}}}(\mathcal{C})$ . Note that  $\sigma(x) \in \mathcal{Val}$  for all variables  $x \in \mathcal{LVar}(\rho')$ . Therefore  $\mathcal{F}_{\text{un}_{\text{te}}}(\ell') = \mathcal{F}_{\text{un}_{\text{te}}}(\ell'\sigma) = \mathcal{F}_{\text{un}}(\ell) \setminus \mathcal{F}_{\text{th}} \subseteq \mathcal{F}_{\text{un}}(\bar{\mathcal{C}}) \setminus \mathcal{F}_{\text{th}} = \mathcal{F}_{\text{un}_{\text{te}}}(\mathcal{C})$ . It remains to show that  $\ell \rightarrow_{\bar{\mathcal{C}}}^* r$ . Our assumption yields  $\ell' \approx r' [\varphi'] \xrightarrow{\mathcal{C}, \geq 1}^* u \approx v [\psi]$  for some trivial  $u \approx v [\psi]$ . From [7, Lemma 2] we obtain a substitution  $\gamma \models \psi$  such that  $\ell'\sigma \rightarrow_{\bar{\mathcal{C}}}^* u\gamma$  and  $r'\sigma = v\gamma$ . Triviality yields  $v\gamma = u\gamma$ . We have  $\rightarrow_{\mathcal{C}} = \rightarrow_{\bar{\mathcal{C}}}$  by [7, Lemma 1] and therefore  $\ell = \ell'\sigma \rightarrow_{\bar{\mathcal{C}}}^* u\gamma = r$ .  $\square$

**Corollary 16.** *If  $\mathcal{R}|_{\mathcal{C}}$  is simulated by  $\mathcal{C}$ ,  $\mathcal{C} \sqsubseteq \mathcal{R}$  and  $\mathcal{R}$  is confluent then  $\mathcal{C}$  is confluent.*

*Proof.* Lemma 15 yields  $\bar{\mathcal{R}}|_{\bar{\mathcal{C}}} \subseteq \rightarrow_{\bar{\mathcal{C}}}^*$ . From  $\rightarrow_{\mathcal{R}} = \rightarrow_{\bar{\mathcal{R}}}$  we obtain the confluence of  $\bar{\mathcal{R}}$ . Hence  $\bar{\mathcal{C}}$  is confluent by Theorem 12. Using  $\rightarrow_{\mathcal{C}} = \rightarrow_{\bar{\mathcal{C}}}$  we obtain the confluence of  $\mathcal{C}$ .  $\square$

**Corollary 17.** *If a left-linear LCTRS  $\mathcal{R}$  is convertible by an LCTRS  $\mathcal{C}$  and  $\mathcal{R}|_{\mathcal{C}}$  is simulated by  $\mathcal{C}$  then  $\mathcal{R}$  is confluent if and only if  $\mathcal{C}$  is confluent.*  $\square$

We have shown that the reduction method works with one particular compositional confluence method. In [8] several additional methods are discussed. We expect that these can also be lifted to the constrained setting.

**Example 18.** Consider the LCTRS  $\mathcal{R}$  of Example 6. An obvious choice for  $\mathcal{C}$  in order to apply Corollary 17 are the rules  $\{\alpha, \gamma, \delta\}$  assuming  $\mathcal{C} \sqsubseteq \mathcal{R}$ . Clearly  $\mathcal{R}$  is left-linear and in Example 10 we have seen that  $\mathcal{R}$  is convertible by  $\mathcal{C}$ . However,  $\mathcal{R}|_{\mathcal{C}}$  is not simulated by  $\mathcal{C}$  and there does not exist any other  $\mathcal{C}$  with  $\mathcal{C} \sqsubseteq \mathcal{R}$  satisfying this.

Comparing Theorem 12 and Corollary 16, a natural question is whether the condition  $\mathcal{C} \sqsubseteq \mathcal{R}$  in the latter can be weakened to  $\rightarrow_{\mathcal{C}}^* \subseteq \rightarrow_{\mathcal{R}}^*$  together with the assumption that  $\mathcal{C}$  and  $\mathcal{R}$  share the same theory. We leave this as future work. Extending *crest* with the results in this section is another topic for future work.

## References

- [1] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. doi:10.1017/CB09781139172752.
- [2] Nao Hirokawa, Julian Nagele, Vincent van Oostrom, and Michio Oyamauchi. Confluence by critical pair analysis revisited. In *Proceedings of the 27th CADE*, volume 11716 of *LNAI*, pages 319–336, 2019. doi:10.1007/978-3-030-29436-6\_19.
- [3] Cynthia Kop and Naoki Nishida. Term rewriting with logical constraints. In *Proceedings of the 9th FroCoS*, volume 8152 of *LNAI*, pages 343–358, 2013. doi:10.1007/978-3-642-40885-4\_24.
- [4] Julian Nagele, Bertram Felgenhauer, and Aart Middeldorp. Improving automatic confluence analysis of rewrite systems by redundant rules. In *Proceedings of the 26th RTA*, volume 36 of *LIPIcs*, pages 257–268, 2015. doi:10.4230/LIPIcs.RTA.2015.257.
- [5] Jonas Schöpfung and Aart Middeldorp. Confluence criteria for logically constrained rewrite systems. In *Proceedings of the 29th CADE*, volume 14132 of *LNAI*, pages 474–490, 2023. doi:10.1007/978-3-031-38499-8\_27.
- [6] Jonas Schöpfung and Aart Middeldorp. Automated analysis of logically constrained rewrite systems using *crest*. In *Proceedings of the 31st TACAS*, volume 15696 of *LNCS*, pages 124–144, 2025. doi:10.1007/978-3-031-90643-5\_7.
- [7] Jonas Schöpfung, Fabian Mitterwallner, and Aart Middeldorp. Confluence of logically constrained rewrite systems revisited. In *Proceedings of the 12th IJCAR*, volume 14740 of *LNAI*, pages 298–316, 2024. doi:10.1007/978-3-031-63501-4\_16.

- [8] Kiraku Shintani and Nao Hirokawa. Compositional confluence criteria. *Logical Methods in Computer Science*, 20(1), 2024. [doi:10.46298/lmcs-20\(1:6\)2024](https://doi.org/10.46298/lmcs-20(1:6)2024).
- [9] Sarah Winkler and Aart Middeldorp. Completion for logically constrained rewriting. In *Proceedings of the 3rd FSCD*, volume 108 of *LIPICs*, pages 30:1–30:18, 2018. [doi:10.4230/LIPICs.FSCD.2018.30](https://doi.org/10.4230/LIPICs.FSCD.2018.30).



# Towards Confluence of Deterministic Higher-Order Pattern Rewrite Systems by Critical Pairs

Johannes Niederhauser and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Innsbruck, Austria  
`{johannes.niederhauser,aart.middeldorp}@uibk.ac.at`

## Abstract

We generalize the concept of critical pairs from Nipkow’s pattern rewrite systems to higher-order rewrite systems where the left-hand sides of rules can be deterministic higher-order patterns.

## 1 Introduction

The goal of this paper is to establish a more general critical pair lemma (Lemma 2) for higher-order rewrite systems (HRSs) à la Nipkow [11]. In [8], such a result is given for pattern rewrite systems (PRSs) which are a restriction of HRSs such that the rules’ left-hand sides belong to a class of lambda terms called patterns which was put forward by Miller [9]. Unlike general higher-order unification, which is undecidable and not unitary [12], unification of patterns is decidable and produces a most general unifier if successful [9]. Furthermore, rewriting with pattern rewrite rules cannot introduce variables [8], which is essential for a well-behaved rewrite system in particular if it should be terminating. While PRSs are favorable from these points of view, they are considerably less expressive than HRSs. In order to alleviate this trade-off, we define in Section 3 a class of HRSs based on deterministic higher-order patterns (DHP) [14] which strictly includes PRSs. We call the resulting formalism deterministic higher-order pattern rewrite systems (DPRSs). All example DPRSs given in this paper go beyond the expressive power of PRSs. For DHPs we still get a deterministic matching algorithm, but neither decidability nor finiteness of the corresponding unification problem has been established to the best of our knowledge. There is the closely related class of functions-as-constructors higher-order unification (FCU) which is decidable and unitary [7]. However, we will show why the imposed restrictions are not practical for critical pair computation in Example 3. Hence, the main result of this paper (Theorem 1) yields a decision procedure for confluence of terminating DPRSs if unification of DHPs is decidable and finite, which we want to settle in future work.

## 2 Preliminaries

Given a binary relation  $R$ ,  $R^+$  and  $R^*$  denote its transitive and transitive-reflexive closure, respectively. For a binary relation  $\rightarrow$  we write  $\leftarrow$ ,  $\leftrightarrow$ ,  $\rightarrow^*$  for its inverse, symmetric closure and transitive-reflexive closure, respectively. In that case, we also define the *joinability relation*  $\downarrow = \rightarrow^* \cdot \leftarrow$ . Throughout this text, we will denote a sequence  $a_1, \dots, a_n$  by  $\overline{a_n}$  where  $n \geq 0$  and the corresponding set by  $\{\overline{a_n}\}$ . For every  $a$ ,  $\overline{a_0}$  represents the empty sequence which we denote by  $()$ . We implicitly enclose sequences in parentheses whenever it is necessary. For a binary relation  $R$ ,  $\overline{a_n} R \overline{b_n}$  abbreviates  $a_1 R b_1, \dots, a_n R b_n$ . Similarly,  $f(\overline{a_n})$  denotes the pointwise application of the function  $f$  to  $\overline{a_n}$ . Given two sequences  $\overline{a_n}$  and  $\overline{b_m}$ , their concatenation is written as  $\overline{a_n}, \overline{b_m}$ .

In this paper, we consider Nipkow's HRSs [8] which operate on simply-typed lambda terms [1, 2]. Let  $\mathcal{S}$  be a set of *sorts*  $(a, b)$ . We use a flattened representation of simple types, so the set  $\mathcal{T}$  of *types*  $(\sigma, \tau)$  is defined as follows:  $\mathcal{S} \subseteq \mathcal{T}$  and if  $\sigma_1, \dots, \sigma_n \in \mathcal{T}$  and  $s \in \mathcal{S}$  then  $\overline{\sigma_n} \rightarrow s \in \mathcal{T}$  where we identify  $() \rightarrow s$  with  $s$ . Since HRSs only work on lambda terms in  $\beta\eta$ -long normal form, we will use a  $\beta\eta$ -free formulation of terms and substitutions based on [3, 4]. This way, we never have to speak about  $\beta$ -reduction,  $\eta$ -reduction and  $\eta$ -expansion in rewriting as every term is in its canonical form at all times. Suppose there is an infinite set  $\mathcal{V}$  of typed variables  $(x, y, z, w)$  and a set  $\mathcal{F}$  of typed function symbols  $(c, d, f, g)$ , such that  $\mathcal{V}, \mathcal{F}, \mathcal{S}$  and  $\{\rightarrow\}$  are disjoint and there are infinitely many variables of each type. A *head*  $h$  is either a function symbol or a variable and we write  $h : \sigma$  to denote that it has type  $\sigma$ . The following inference rules define the set  $\text{Term}(\mathcal{F}, \mathcal{V}, \sigma)$  of *terms*  $(s, t, u, v)$  of type  $\sigma$

$$\frac{h : \overline{\sigma_n} \rightarrow a \in \mathcal{F} \cup \mathcal{V} \quad \overline{t_n} \in \text{Term}(\mathcal{F}, \mathcal{V}, \overline{\sigma_n})}{h(\overline{t_n}) \in \text{Term}(\mathcal{F}, \mathcal{V}, a)} \quad \frac{t \in \text{Term}(\mathcal{F}, \mathcal{V}, a) \quad \overline{x_n} : \overline{\sigma_n} \in \mathcal{V}^n}{\overline{x_n}.t \in \text{Term}(\mathcal{F}, \mathcal{V}, \overline{\sigma_n} \rightarrow a)}$$

where we use  $\text{Term}(\mathcal{F}, \mathcal{V}, \overline{\sigma_n})$  as a shorthand for  $\text{Term}(\mathcal{F}, \mathcal{V}, \sigma_1) \times \dots \times \text{Term}(\mathcal{F}, \mathcal{V}, \sigma_n)$ . We abbreviate  $h()$  by  $h$  and follow the convention that postfix operations  $\diamond$  bind stronger than binders in terms, so  $\overline{x_n}.t \diamond = \overline{x_n}.(t \diamond)$ . The set  $\text{Term}(\mathcal{F}, \mathcal{V}, \mathcal{S})$  denotes the set of all terms which coincides with the set of well-typed lambda terms over  $\mathcal{F}$  and  $\mathcal{V}$  in  $\beta\eta$ -long normal form. As for function symbols and variables, we denote that a term  $s$  has type  $\sigma$  by  $s : \sigma$ . Given a term  $s$ , its set of free ( $\text{FV}(s)$ ) and bound variables ( $\text{BV}(s)$ ) are defined as usual.

We view terms modulo renaming of bound variables ( $\alpha$ -renaming). Hence, we may assume that no variable occurs both free and bound in any term. Note that  $\mathcal{V} \subseteq \text{Term}(\mathcal{F}, \mathcal{V}, \mathcal{S})$  does not hold as terms are always in canonical form. The function  $\uparrow$  takes a variable and returns its canonical form by performing  $\eta$ -expansion:  $x \uparrow = \overline{y_n}.x(\overline{y_n} \uparrow)$  whenever  $x : \overline{\sigma_n} \rightarrow a$ ,  $\overline{y_n} : \overline{\sigma_n}$  and  $x \notin \{\overline{y_n}\}$ .

Finite mappings from variables to terms of the same type are called *substitutions*  $(\theta, \gamma, \delta, \mu)$ . Given a substitution  $\theta = \{\overline{x_n} \mapsto \overline{t_n}\}$ , its *domain* and *image* are defined as  $\text{Dom}(\theta) = \{\overline{x_n}\}$  and  $\text{Im}(\theta) = \{\overline{t_n}\}$ , respectively. The *free variables* introduced by a substitution  $\theta$  are defined as the set  $\text{FV}(\theta) = \bigcup \{\text{FV}(t) \mid t \in \text{Im}(\theta)\}$ . A substitution is called a *renaming* if  $\text{Im}(\theta) \subseteq \{x \uparrow \mid x \in \mathcal{V}\}$  and the corresponding mapping from  $\mathcal{V}$  to  $\mathcal{V}$  is bijective. We often refer to bijective mappings from  $\mathcal{V}$  to  $\mathcal{V}$  as *variable renamings*. Since terms are always in their canonical form, the application of a substitution  $\theta$  to a term  $t$  (written in postfix notation  $t\theta$ ) is defined hereditarily by implicitly performing  $\beta$ -reduction [6]:  $x(\overline{t_n})\theta = u\{\overline{x_n} \mapsto \overline{t_n}\theta\}$  if  $x \mapsto \overline{x_n}.u \in \theta$ ,  $h(\overline{t_n})\theta = h(\overline{t_n}\theta)$  if  $h \notin \text{Dom}(\theta)$  and  $(\overline{x_n}.t)\theta = \overline{x_n}.t\theta$ . Note that we always assume that  $(\text{Dom}(\theta) \cup \text{FV}(\theta)) \cap \text{BV}(t) = \emptyset$  by employing  $\alpha$ -renaming. Hence,  $t\theta$  represents the capture-avoiding application of  $\theta$  to the free variables of  $t$ . We say that a term  $s$  *matches* a term  $t$  if there exists a substitution  $\theta$  such that  $s\theta = t$ . Moreover, if  $s\theta = t\theta$  then  $\theta$  is a *unifier* of  $s$  and  $t$ . Given  $\theta = \{\overline{x_n} \mapsto \overline{s_n}\}$  and  $\delta = \{\overline{y_m} \mapsto \overline{t_m}\}$ , we define their composition as  $\theta\delta = \{\overline{x_n} \mapsto \overline{s_n}\delta\} \cup \{\overline{y_j} \mapsto \overline{t_j} \mid 1 \leq j \leq m \text{ and } \overline{y_j} \notin \{\overline{x_n}\}\}$ . A substitution  $\theta$  is *at least as general* as a substitution  $\delta$ , denoted by  $\theta \leq_\beta \delta$ , if there exists a substitution  $\mu$  such that  $\theta\mu = \delta$ .

Given a term  $s$ , a sequence of terms  $\overline{x_n}$  and a set of variables  $W$  we define the  $\overline{x_n}$ -lifter of  $s$  away from  $W$  to be the substitution  $\{x \mapsto \overline{y_m}.\rho(x)(\overline{x_n}, \overline{y_m}) \mid x : \overline{\tau_m} \rightarrow a \in \text{FV}(s)\}$  where  $\rho$  is a variable renaming away from  $W$ ,  $\rho(x) : (\overline{\sigma_n}, \overline{\tau_m}) \rightarrow a$ ,  $\overline{y_m} : \overline{\tau_m}$  and  $\overline{x_n} : \overline{\sigma_n}$ .

A *position*  $(p, q, r)$  is a string over  $\mathbb{N}$  where  $\epsilon$  denotes the empty string which is often called the *root position*. We write  $pq$  for the concatenation of two positions  $p$  and  $q$  and  $p \leq q$  if  $p$  is a prefix of  $q$ , i.e., there exists a position  $r$  such that  $pr = q$ . In that case, we say that  $p$  is *above*  $q$  and  $q$  is *below*  $p$ . If neither  $p \leq q$  nor  $q \leq p$  then  $p$  and  $q$  are *parallel*, written  $p \parallel q$ . The set  $\text{Pos}(s)$  contains all positions of a term  $s$ . Given a position  $p \in \text{Pos}(s)$ ,  $s|_p$  denotes the

*abstracted subterm* at position  $p$  which is defined inductively as follows:  $s|_\epsilon = s$ ,  $h(\overline{t_n})|_{ip} = t_i|_p$  and  $(\overline{x_n}.t)|_p = \overline{x_n}.t|_p$ . In particular, the scope of bound variables is not discarded, so the definition of abstracted subterms is well-defined in the presence of  $\alpha$ -renaming. If  $s|_p = \overline{x_n}.t$  we refer to  $\overline{x_n}$  as  $\text{bv}(s, p)$ . Furthermore, we use  $s \supseteq t$  to denote that  $t$  is an abstracted subterm of  $s$  and define  $s \triangleright t$  as  $s \supseteq t$  and  $s \neq t$ . A position  $p \in \text{Pos}(s)$  is called *functional* (denoted by  $p \in \text{Pos}_{\mathcal{F}}(s)$ ) if the head of  $s|_p$  is a function symbol. Finally, given  $p \in \text{Pos}(s)$  and a term  $\overline{x_n}.t$  of the same type as  $s|_p = \overline{x_n}.u$  we define  $s[\overline{x_n}.t]_p$  as the result of replacing  $u$  by  $t$  in  $s$ . Note that this would not be well-defined if we chose  $\beta\eta$ -normal forms as our canonical forms due to the possible removal of variable occurrences through subterm replacement: For  $s = x.f(h(x), x)$ ,  $s[x.c]_1 = x.f(c, x)$  is  $\eta$ -reducible.

A pair of terms  $(\ell, r)$  with  $\ell, r : a$  can be seen as an *equation* (written  $\ell \approx r$ ), or, if we additionally assume that  $\text{FV}(r) \subseteq \text{FV}(\ell)$  and the head of  $\ell$  is not a variable, as a *rule* (written  $\ell \rightarrow r$ ). A *higher-order equational system* (HES) is a set of equations while a *higher-order rewrite system* (HRS) is a set of rules. Every rule can be seen as an equation, so every HRS is also an HES. We will exploit this by giving some general definitions only for HESs. Given an HES  $\mathcal{E}$ , its *rewrite relation*  $\rightarrow_{\mathcal{E}}$  is defined as follows: There is a *rewrite step*  $s \rightarrow_{\mathcal{E}} t$  if there exist an equation  $\ell \approx r \in \mathcal{E}$ , a substitution  $\theta$  and a position  $p \in \text{Pos}(s)$  such that  $s|_p = \overline{x_n}.\ell\theta$  and  $t = s[\overline{x_n}.r\theta]_p$ . Sometimes, we make the position  $p$  explicit by writing  $s \rightarrow_{\mathcal{E}}^p t$ . Two equations  $\ell_1 \approx r_1$  and  $\ell_2 \approx r_2$  are *variants* if there exists a renaming  $\gamma$  such that  $\ell_1\gamma = \ell_2$  and  $r_1\gamma = r_2$ . Our definition of HRSs is equivalent to the original one given in [8]. Finally, we say that an HRS  $\mathcal{R}$  is *terminating* if  $\rightarrow_{\mathcal{R}}$  is well-founded and call  $\mathcal{R}$  (*locally*) *confluent* if  $\mathcal{R}^* \leftarrow \cdot \rightarrow^* \mathcal{R} \subseteq \downarrow_{\mathcal{R}}$  ( $\mathcal{R}^* \leftarrow \cdot \rightarrow^* \mathcal{R} \subseteq \downarrow_{\mathcal{R}}$ ).

### 3 Deterministic Higher-Order Patterns

DHPs as introduced in [14] come with a deterministic matching problem as well as an algorithm which computes matching substitutions in linear time. In order to simplify the presentation of the following definition, we will resort to the usage of  $\beta\eta$ -normal forms. To that end,  $s\downarrow_{\eta}$  denotes the  $\eta$ -normal form of a term  $s$  as a lambda term (employing the usual applicative notation). Furthermore, let  $\triangleright_{\text{ho}}$  be the higher-order subterm relation on lambda terms. (The main difference from our subterm relation  $\triangleright$  is that  $g(x) \not\triangleright g$  but  $g x \triangleright_{\text{ho}} g$ .)

**Definition 1.** A term  $s$  is a *deterministic higher-order pattern* (DHP) if the following conditions hold for all abstracted subterms  $\overline{y_n}.x(\overline{t_m})$  with  $x \notin \{\overline{y_n}\}$  and  $1 \leq i \leq m$ :  $\emptyset \neq \text{FV}(t_i) \subseteq \{\overline{y_n}\}$ ,  $t_i\downarrow_{\eta} \not\triangleright_{\text{ho}} t_j\downarrow_{\eta}$  whenever  $i \neq j$ , and  $t_i\downarrow_{\eta}$  is not a lambda abstraction.

We are now ready to define a subclass of HRSs for which we will be able to obtain a critical pair lemma. A *deterministic higher-order pattern rewrite rule* is a rewrite rule whose left-hand side is a DHP. A *deterministic higher-order pattern rewrite system* (DPRS) is a set of deterministic higher-order pattern rewrite rules. To the best of our knowledge, DPRSs have not been considered in the literature. Note that like for PRSs, rewriting with DPRSs cannot introduce variables since free variables are not nested in their left-hand sides. As opposed to Miller's pattern unification [9] or FCU [7], unification of DHPs is not unitary.

**Example 1** (taken from [14]). Consider the sort  $a$ , the function symbol  $f : a \rightarrow a$  as well as the variables  $M, N : (a, a) \rightarrow a$ . The terms  $x, y.M(f(x), f(y))$  and  $x, y.f(N(y, x))$  admit the three unifiers  $\{M \mapsto z_1, z_2.z_1, N \mapsto z_1, z_2.z_2\}$ ,  $\{M \mapsto z_1, z_2.z_2, N \mapsto z_1, z_2.z_1\}$  and  $\{M \mapsto z_1, z_2.f(Z(z_1, z_2)), N \mapsto z_1, z_2.Z(f(z_2), f(z_1))\}$  where  $Z : (a, a) \rightarrow a$  is a fresh variable. Note that all three unifiers are incomparable with respect to  $\leq_{\beta}$ .

A *minimal complete set of unifiers*  $U$  of two terms  $s$  and  $t$  satisfies the following conditions: For all  $\theta \in U$ ,  $\theta$  is a unifier of  $s$  and  $t$ . Furthermore, if  $\delta$  is a unifier of  $s$  and  $t$  then  $\theta \leqslant_\beta \delta$  for some  $\theta \in U$ . Finally, the elements of  $U$  are incomparable with respect to  $\leqslant_\beta$ . Assuming that minimal sets of unifiers of DHPs are finite and computable, the critical pair lemma stated in the following section yields a decision procedure for confluence of terminating DPRSs.

## 4 Critical Peaks

In DPRSs, the free variables in left-hand sides of rules may be applied to arguments which contain function symbols. Therefore, as opposed to higher-order rewriting with PRSs, in addition to overlaps at function positions, overlaps at variable positions also have to be considered. To the best of our knowledge, Hamana [5] provides the sole existing definition of critical pairs in such a setting, albeit for the restricted class of second-order systems. The main challenge of establishing a critical pair lemma for DPRSs is to give a complete characterization of overlaps at variable positions using only a finite number of critical pairs.

**Example 2.** Consider the sorts  $a$  and  $b$ , the function symbols  $c : (a \rightarrow a) \rightarrow a$ ,  $d : a$ ,  $e : b \rightarrow a$ ,  $f, g : a \rightarrow a$ ,  $h : a \rightarrow b$  as well as the variables  $x : a$  and  $Z : a \rightarrow a$ . The DPRS  $\mathcal{R}$  consisting of the following rules

$$f(g(x)) \rightarrow f(x) \qquad h(g(x)) \rightarrow h(x) \qquad c(y.Z(g(y))) \rightarrow Z(d)$$

exhibits the local peak  $c(y.f(y)) \xrightarrow{\mathcal{R}} c(y.f(g(y))) \xrightarrow{\epsilon} f(d)$  which is not joinable. Following [5] we could consider  $c(y.f(y)) \approx f(d)$  to be a critical pair. However, the involved rules may also overlap at some lower position in a given instance of  $Z$ , so in addition we have the non-joinable local peak  $c(y.f(f(y))) \xrightarrow{\mathcal{R}} c(y.f(f(g(y)))) \xrightarrow{\epsilon} f(f(d))$ . Hence, given a fresh variable  $Y' : a \rightarrow a$  which can be instantiated by a function representing the concrete context, the choice of  $c(y.Y'(f(y))) \approx Y'(f(d))$  as critical pair seems to be more apt. Furthermore, considering a variable  $F : (a, a) \rightarrow a$ , the local peak  $c(y.F(f(y), g(y))) \xrightarrow{\mathcal{R}} c(y.F(f(g(y)), g(y))) \xrightarrow{\epsilon} F(f(d), d)$  is not captured by this critical pair as  $c(y.Y'(f(y)))\{Y' \mapsto z.F(z, g(y))\} = c(x.F(f(x), g(y)))$ . Hence, the original arguments of  $Z$  should be added to the arguments of the fresh variable introduced in the critical pair. Using the fresh variable  $Z' : (a, a) \rightarrow a$ ,  $c(y.Z'(f(y), g(y))) \approx Z'(f(d), d)$  characterizes all non-joinable local peaks considered so far. By utilizing this strategy of using a fresh variable as a placeholder for the context up to the actual overlap, we can now also provide the critical pair  $c(y.Z''(h(y), g(y))) \approx Z''(h(d), d)$  where  $Z'' : (b, a) \rightarrow a$  is a fresh variable for the non-joinable local peak  $c(y.e(h(y))) \xrightarrow{\mathcal{R}} c(y.e(h(g(y)))) \xrightarrow{\epsilon} e(h(d))$  even though  $h(g(x))$  and  $Z(g(x))$  have different types. Note that despite  $\mathcal{R}$  being a second-order system, all these crucial refinements of the naive notion of critical pairs given initially are not present in [5], rendering its local confluence result (Theorem 7.11) unsound: According to the definition given there,  $c(y.f(y)) \approx f(d)$  is the only critical pair of  $\mathcal{R}$ . Following the same definition, the DPRS  $\mathcal{R}' = \mathcal{R} \cup \{c(y.f(y)) \rightarrow f(d)\}$  still only has one critical pair which is now joinable. Hence, according to [5],  $\mathcal{R}'$  is locally confluent but actually this is not the case as the local peak  $c(y.f(f(y))) \xrightarrow{\mathcal{R}'} c(y.f(f(g(y)))) \xrightarrow{\epsilon} f(f(d))$  is still not joinable.

In the following, the considerations of the previous example are transformed into a formal definition of critical pairs for DPRSs. We start by giving a definition of overlaps. For function positions, we just have to adapt the the definition from PRSs for minimal complete sets of unifiers instead of most general unifiers. For variable positions, we have to account for overlaps which may occur at a lower position and therefore also at a different sort than the output sort

of the variable under consideration. However, such overlaps only have to be considered when an argument of the free variable which is not merely the canonical representation of a bound variable is used in the unifier.

**Definition 2.** An *overlap* of a DPRS  $\mathcal{R}$  is an octuple  $\langle \ell_1 \rightarrow r_1, p, q, \overline{x_n}, \delta, \gamma, U, \ell_2 \rightarrow r_2 \rangle$  satisfying the following properties:

- (i)  $\ell_1 \rightarrow r_1, \ell_2 \rightarrow r_2 \in \mathcal{R}$ ,
- (ii)  $p \in \mathcal{Pos}(\ell_2)$ ,  $\text{bv}(\ell_2, p) = \overline{x_n}$  and  $\delta$  is an  $\overline{x_n}$ -lifter of  $\ell_1$  away from  $\text{FV}(\ell_2)$ ,
- (iii)  $U'$  is a minimal complete set of unifiers of  $\overline{x_n}.\ell_1\delta$  and  $\ell_2\gamma|_{pq}$ ,
- (iv) either (a)  $q = \epsilon$ ,  $\gamma = \emptyset$ ,  $p \in \mathcal{Pos}_{\mathcal{F}}(\ell_2)$  and  $U = U'$ , or (b)  $q = 1$  and
  - $\ell_2|_p = \overline{x_n}.y(\overline{s_m})$  where  $y \notin \{\overline{x_n}\}$  and  $\{\overline{s_m}\} \not\subseteq \{\overline{x_n}\uparrow\}$ ,
  - $\gamma = \{y \mapsto \overline{y_m}.y''(y'(\overline{y_m}\uparrow), \overline{y_m}\uparrow)\}$  where  $y' : \overline{\sigma_m} \rightarrow b$  and  $y'' : (b, \overline{\sigma_m}) \rightarrow a$  are fresh variables assuming  $y : \overline{\sigma_m} \rightarrow a$  and  $\ell_1 : b$
  - $U = U' \setminus U''$  such that for all  $\theta \in U'' \subseteq U$ , at least one of the following holds:
    - if  $\theta(y') = \overline{y_m}.v$  then  $y_i \in \text{FV}(v)$  implies  $s_i \in \{\overline{x_n}\uparrow\}$  for all  $i$ ,
    - $(\overline{x_n}.y'(\overline{s_m}))\theta \in \{\overline{x_n}.s_i \mid 1 \leq i \leq m\}$
- (v) if  $p = \epsilon$  then  $\ell_1 \rightarrow r_1$  and  $\ell_2 \rightarrow r_2$  are not variants

Note that the second condition stated in the third item of property (iv)(b) guarantees that overlaps at arguments of free variables are not considered as overlaps at these free variables. The following example clarifies the differences from the previous definition with [5, Definition 7.7] which have not already been addressed.

**Example 3.** Consider the sort  $a$ , the function symbols  $c : (a \rightarrow a) \rightarrow a$ ,  $d : a$ ,  $f, g, h : a \rightarrow a$  as well as the variables  $x : a$  and  $Z : (a, a) \rightarrow a$ . The DPRS  $\mathcal{R}$  consisting of the following rules

$$f(g(x)) \rightarrow x \qquad c(y.Z(g(y), h(y))) \rightarrow Z(d, d)$$

contains the overlap  $\langle f(g(x)) \rightarrow x, 1, 1, y, \delta, \gamma, U, c(y.Z(g(y), h(y))) \rightarrow Z(d, d) \rangle$  with  $\delta = \{x \mapsto X'(y)\}$  and  $\gamma = \{Z \rightarrow z_1, z_2.Z''(Z'(z_1, z_2), z_1, z_2)\}$  where  $X' : a \rightarrow a$ ,  $Z' : (a, a) \rightarrow a$  and  $Z'' : (a, a, a) \rightarrow a$  are fresh variables. For  $\theta = \{X' \mapsto z.z, Z' \mapsto z_1, z_2.f(z_1)\} \in U$  and the instance where we map  $Z''$  to  $z.z$ , this overlap gives rise to the non-joinable local peak  $c(y.y) \xrightarrow{\mathcal{R}} c(y.f(g(y))) \xrightarrow{\epsilon} c(y.f(d))$ . Note that since  $z_2 \notin \text{FV}(f(z_1))$ , this would not be an overlap according to [5, Definition 7.7]. Furthermore, we can see why FCU is not enough for our purposes: The unification problem  $y.f(g(X'(y))) \approx y.Z'(g(y), h(y))$  does not satisfy the global restriction of the FCU class as  $g(y) \triangleright y$ . In [5], a solution for second-order systems is provided by extending the FCU algorithm to this special case. Note that due to the usage of a lifter  $\delta$ , ordinary FCU cannot be used in many cases. In particular, even though lifters are not used in [5], we cannot dispense of them: The unification problem  $y.f(g(x)) \approx y.Z'(g(y), h(y))$  has no solutions as we may not set  $\theta(x) = y$  due to capture-avoidance. Furthermore, we want to abstract over  $y$  since it prevents  $y$  from being in the domain of  $\theta$ .

The upcoming result ensures that there exists a source term for overlaps. After that, we transform overlaps into critical peaks which leads to the key definition of critical pairs.

**Lemma 1.** *If  $\langle \ell_1 \rightarrow r_1, p, q, \overline{x_n}, \delta, \gamma, U, \ell_2 \rightarrow r_2 \rangle$  is an overlap of  $\mathcal{R}$  then  $\ell_2\gamma\theta[(\overline{x_n}.\ell_1\delta)\theta]_{pq} = \ell_2\gamma\theta$  for all  $\theta \in U$ .*

**Definition 3.** Let  $\langle \ell_1 \rightarrow r_1, p, q, \overline{x_n}, \delta, \gamma, U, \ell_2 \rightarrow r_2 \rangle$  be an overlap of a DPRS  $\mathcal{R}$  and  $\theta \in U$ . By Lemma 1,  $\ell_2 \gamma \theta[(\overline{x_n} \cdot \ell_1 \delta) \theta]_{pq} = \ell_2 \gamma \theta$ . This term can be rewritten in two ways

$$\ell_2 \gamma \theta[(\overline{x_n} \cdot r_1 \delta) \theta]_{pq} \xrightarrow{pq|\ell_1 \rightarrow r_1}_{\mathcal{R}} \ell_2 \gamma \theta[(\overline{x_n} \cdot \ell_1 \delta) \theta]_{pq} = \ell_2 \gamma \theta \xrightarrow{\epsilon|\ell_2 \rightarrow r_2}_{\mathcal{R}} r_2 \gamma \theta$$

which gives rise to a *critical peak* of  $\mathcal{R}$ . The equation  $\ell_2 \gamma \theta[(\overline{x_n} \cdot r_1 \delta) \theta]_{pq} \approx r_2 \gamma \theta$  is called a *critical pair* of  $\mathcal{R}$ . The set of critical pairs is denoted by  $\text{CP}(\mathcal{R})$ .

**Example 4.** Consider the sort **prop**, the function symbols  $\perp, \top : \text{prop}$ ,  $\neg : \text{prop} \rightarrow \text{prop}$ ,  $\wedge, \vee, \Rightarrow : (\text{prop}, \text{prop}) \rightarrow \text{prop}$ ,  $\text{repl} : (\text{prop} \rightarrow \text{prop}, \text{prop}) \rightarrow \text{prop}$  as well as the variables  $x : \text{prop}$  and  $F : \text{prop} \rightarrow \text{prop}$ . The DPRS  $\mathcal{R}$  consisting of the following rules

$$\begin{array}{lll} \text{repl}(y.F(\neg y), x) \rightarrow F(x \Rightarrow \perp) & \neg \neg x \rightarrow x & \neg(x \Rightarrow \perp) \rightarrow x \\ \text{repl}(y.F(y), x) \rightarrow F(x) & \neg x \rightarrow x \Rightarrow \perp & (x \Rightarrow \perp) \Rightarrow \perp \rightarrow x \end{array}$$

makes use of DHPs to facilitate replacement of  $\neg x$  by  $x \Rightarrow \perp$  for arbitrary  $x$  in just one rewrite step through  $\text{repl}$  which is not possible with PRSs. Let  $H_1 : \text{prop} \rightarrow \text{prop}$ ,  $H_2 : (\text{prop}, \text{prop}) \rightarrow \text{prop}$  and  $G : (\text{prop}, \text{prop}) \rightarrow \text{prop}$  be fresh variables. Modulo symmetry, the system admits one critical pair  $H_1(x \Rightarrow \perp) \approx H_1(\neg x)$  from the root overlap of the two rules defining  $\text{repl}$ . Moreover, there are variable overlaps between the first rule defining  $\text{repl}$  and itself as well as the second rule defining  $\text{repl}$  resulting in the following critical pairs:

$$\begin{array}{l} \text{repl}(y.G(H_2(\neg y, H_1(\neg y) \Rightarrow \perp), \neg y), x) \approx G(\text{repl}(z.H_2(x \Rightarrow \perp, \neg z), H_1(x \Rightarrow \perp)), x \Rightarrow \perp) \\ \text{repl}(y.G(H_2(\neg y, H_1(\neg y)), \neg y), x) \approx G(\text{repl}(z.H_2(x \Rightarrow \perp, z), H_1(x \Rightarrow \perp)), x \Rightarrow \perp) \end{array}$$

Furthermore, the first-order part gives rise to the critical pairs  $\neg x \approx \neg x$ ,  $\neg x \Rightarrow \perp \approx x$ ,  $\neg(x \Rightarrow \perp) \approx x$ ,  $\neg x \approx x \Rightarrow \perp$ ,  $(x \Rightarrow \perp) \Rightarrow \perp \approx x$  and  $x \Rightarrow \perp \approx x \Rightarrow \perp$  (modulo symmetry). A variable overlap between the first rule defining  $\text{repl}$  and  $\neg \neg x \rightarrow x$  yields the critical pair  $\text{repl}(y.G(y, \neg y), x) \approx G(\neg(x \Rightarrow \perp), x \Rightarrow \perp)$ . Finally,  $\text{repl}(y.F(y \Rightarrow \perp), x) \approx F(x \Rightarrow \perp)$  is obtained from a variable argument overlap between the first rule defining  $\text{repl}$  and  $\neg x \rightarrow x \Rightarrow \perp$ .

With all definitions in place, we can now formulate a critical pair lemma for DPRSs. Note that terms which are not joinable may not be directly connected by a critical pair. However, this does not pose an issue as joinable critical pairs still yield joinability of all local peaks.

**Lemma 2.** Let  $\mathcal{R}$  be a DPRS. If  $t \mathcal{R} \leftarrow \cdot \rightarrow_{\mathcal{R}} u$  then  $t \downarrow_{\mathcal{R}} u$  or  $t \rightarrow_{\mathcal{R}}^* \cdot \leftrightarrow_{\text{CP}(\mathcal{R})} \cdot \mathcal{R}^* \leftarrow u$ .

Finally, we state how Lemma 2 can be used to establish confluence of terminating DPRSs.

**Theorem 1.** A terminating DPRS is confluent if and only if all its critical pairs are joinable.

*Proof.* Let  $\mathcal{R}$  be a DPRS. If all its critical pairs are joinable then  $\leftrightarrow_{\text{CP}(\mathcal{R})} \subseteq \downarrow_{\mathcal{R}}$  and thus  $\mathcal{R} \leftarrow \cdot \rightarrow_{\mathcal{R}} \subseteq \downarrow_{\mathcal{R}}$  by Lemma 2. Hence,  $\mathcal{R}$  is locally confluent. Together with termination of  $\mathcal{R}$  we conclude confluence of  $\mathcal{R}$  by Newman's Lemma [10].  $\square$

**Example 5.** Recall the DPRS  $\mathcal{R}$  from Example 4. The critical pairs which do not contain abstractions are easily joinable. For the other critical pairs between the rules defining  $\text{repl}$  we have  $\text{repl}(y.G(H_2(\neg y, H_1(\neg y) \Rightarrow \perp), \neg y), x) \rightarrow_{\mathcal{R}} G(H_2(x \Rightarrow \perp, H_1(x \Rightarrow \perp) \Rightarrow \perp), x \Rightarrow \perp) \mathcal{R} \leftarrow G(\text{repl}(z.H_2(x \Rightarrow \perp, \neg z), H_1(x \Rightarrow \perp)), x \Rightarrow \perp)$  and  $\text{repl}(y.G(H_2(\neg y, H_1(\neg y)), \neg y), x) \rightarrow_{\mathcal{R}} G(H_2(x \Rightarrow \perp, H_1(x \Rightarrow \perp)), x \Rightarrow \perp) \mathcal{R} \leftarrow G(\text{repl}(z.H_2(x \Rightarrow \perp, z), H_1(x \Rightarrow \perp)), x \Rightarrow \perp)$ . The remaining critical pairs are also joinable:  $\text{repl}(y.G(y, \neg y), x) \rightarrow_{\mathcal{R}} G(x, \neg x)$  using the second rule defining  $\text{repl}$  with the substitution  $\{F \mapsto z.G(z, \neg z)\}$ ,  $G(x, \neg x) \rightarrow_{\mathcal{R}} G(x, x \Rightarrow \perp)$  and  $G(\neg(x \Rightarrow \perp), x \Rightarrow \perp) \rightarrow_{\mathcal{R}} G(x, x \Rightarrow \perp)$ . Similarly,  $\text{repl}(y.F(y \Rightarrow \perp), x) \rightarrow_{\mathcal{R}} F(x \Rightarrow \perp)$ . Termination of  $\mathcal{R}$  can be established by using straightforward interpretations of the function symbols with the monotone algebra approach for HRS put forward by van de Pol [13]. Hence, Theorem 1 yields confluence of  $\mathcal{R}$ .



## References

- [1] Henk P. Barendregt. Lambda calculi with types. In Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press, 1992. doi: [10.1093/oso/9780198537618.003.0002](https://doi.org/10.1093/oso/9780198537618.003.0002).
- [2] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, 1940. doi: [10.2307/2266170](https://doi.org/10.2307/2266170).
- [3] Thiago Felicissimo. A confluence criterion for non left-linearity in a  $\beta\eta$ -free reformulation of HRSs. In Carsten Fuhs, editor, *Proc. 11th International Workshop on Higher-Order Rewriting*, pages 6–11, 2023.
- [4] Thiago Felicissimo and Jean-Pierre Jouannaud. Sort-based confluence criteria for non-left-linear higher-order rewriting. In *Proc. 30th International Conference on Automated Deduction*, 2025. To appear.
- [5] Makoto Hamana. How to prove decidability of equational theories with second-order computation analyser SOL. *Journal of Functional Programming*, 29:e20, 2019. doi: [10.1017/S0956796819000157](https://doi.org/10.1017/S0956796819000157).
- [6] Robert Harper and Daniel R. Licata. Mechanizing metatheory in a logical framework. *Journal of Functional Programming*, 17(4–5):613–673, 2007. doi: [10.1017/S0956796807006430](https://doi.org/10.1017/S0956796807006430).
- [7] Tomer Libal and Dale Miller. Functions-as-constructors higher-order unification: Extended pattern unification. *Annals of Mathematics and Artificial Intelligence*, 90:455–479, 2022. doi: [10.1007/s10472-021-09774-y](https://doi.org/10.1007/s10472-021-09774-y).
- [8] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, 1998. doi: [10.1016/S0304-3975\(97\)00143-6](https://doi.org/10.1016/S0304-3975(97)00143-6).
- [9] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991. doi: [10.1093/log-com/1.4.497](https://doi.org/10.1093/log-com/1.4.497).
- [10] Max H. A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223–243, 1942. doi: [10.2307/1968867](https://doi.org/10.2307/1968867).
- [11] Tobias Nipkow. Higher-order critical pairs. In *Proc. 6th Annual Symposium on Logic In computer Science*, pages 342–349, 1991. doi: [10.1109/LICS.1991.151658](https://doi.org/10.1109/LICS.1991.151658).
- [12] Wayne Snyder and Jean Gallier. Higher-order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8(1–2):101–140, 1989. doi: [10.1016/S0747-7171\(89\)80023-9](https://doi.org/10.1016/S0747-7171(89)80023-9).
- [13] Jaco van de Pol. *Termination of Higher-Order Rewrite Systems*. PhD thesis, Utrecht University, 1996.
- [14] Tetsuo Yokoyama, Zhenjiang Hu, and Masato Takeichi. Deterministic higher-order patterns for program transformation. In Maurice Bruynooghe, editor, *Proc. 13th International Symposium on Logic-Based Program Synthesis and Transformation*, volume 3018 of *Lecture Notes in Computer Science*, pages 128–142, 2004. doi: [10.1007/978-3-540-25938-1\\_12](https://doi.org/10.1007/978-3-540-25938-1_12).

# Term Evaluation Systems with Refinements for Contextual Improvement by Critical Pair Analysis\*

## (Extended Abstract)

Makoto Hamana and Koko Muroya

<sup>1</sup> Department of Computer Science and Networks, Kyushu Institute of Technology, Japan

hamana@csn.kyutech.ac.jp

<sup>2</sup> Ochanomizu University, Japan

kmuroya@is.ocha.ac.jp

### Abstract

For a programming language, there are two kinds of term rewriting: run-time rewriting (“evaluation”) and compile-time rewriting (“refinement”). Whereas refinement resembles general term rewriting, evaluation is commonly constrained by Felleisen’s evaluation contexts. While evaluation specifies a programming language, refinement models optimisation and should be validated with respect to evaluation. Such validation can be given by Sands’ notion of contextual improvement. We formulate evaluation in a term-rewriting-theoretic manner for the first time, and introduce Term Evaluation and Refinement Systems (TERS). We then identify sufficient conditions for contextual improvement, and provide critical pair analysis for local coherence that is the key sufficient condition. As case studies, we prove contextual improvement for a computational lambda-calculus and its extension with effect handlers.

## 1 Introduction

Term rewriting is a general model of computation. The ecosystem of a functional programming language utilizes two types of term rewriting: run-time rewriting, which we shall refer to as *evaluation*, and compile-time rewriting, referred to as *refinement*. Run-time evaluation specifies operational semantics of the language. It can only happen in a particular order, usually deterministically. On the other hand, compile-time refinement models optimisation. It can happen anywhere, nondeterministically. The difference between evaluation and refinement, as kinds of term rewriting, can be summarized in terms of *contexts*:

$$\frac{(l \rightarrow r) \in \mathcal{E} \quad E \in Ectx}{E[l\theta] \rightarrow_{\mathcal{E}} E[r\theta]} \qquad \frac{(l \Rightarrow r) \in \mathcal{R} \quad C \in Ctx}{C[l\theta] \Rightarrow_{\mathcal{R}} C[r\theta]}$$

Evaluation  $\rightarrow_{\mathcal{E}}$  uses a rewrite rule  $l \rightarrow r$  inside a Felleisen’s *evaluation context* [3, 2]  $E \in Ectx$  only; this is a new kind of restriction from the rewriting theoretic point of view. In contrast, refinement  $\Rightarrow_{\mathcal{R}}$  uses a rewrite rule  $l \Rightarrow r$  inside an *arbitrary* context  $C \in Ctx$ ; this resembles general term rewriting.

We analyse the roles of term rewriting in programming languages in this manner and divide them into *evaluation* and *refinement* for formalisation. This constitutes a novel theory that is more suitable as a semantics of programming languages. Evaluation *specifies* (the behavior of) a programming language as operational semantics. Evaluation is not merely a deterministic restriction of refinement. Refinement which models optimisation should be *validated* with

---

\*This is an extended abstract of the paper presented at the FLOPS 2024 [6].



respect to evaluation. Indeed, compiler optimisation is intended to preserve evaluation results and improve time efficiency of evaluation. Such preservation and improvement deserve formal validation.

Such validation can be provided as *observational equivalence* [4], and its quantitative variant, *contextual improvement* [8]. Observational equivalence  $t \cong u$  asserts that two terms  $t$  and  $u$  cannot be distinguished by any context  $C$ ; formally, if  $C[t]$  terminates,  $C[u]$  terminates with the same evaluation result, and vice versa. Contextual improvement additionally asserts that  $C[u]$  terminates with no more evaluation steps than  $C[t]$ . This is a suitable notion to validate refinement which models optimisation.

Whereas the theory of refinement, which resembles general term rewriting, has been deeply developed, evaluation seems to be a new kind of restricted rewriting and it lacks a general theory from the perspective of term rewriting. This prevents useful ideas and techniques of term rewriting from transferring from refinement to evaluation. In recent work [5] on a proof methodology of observational equivalence, it is informally observed that a rewriting technique can be useful for proving observational equivalence and contextual improvement. This methodology informally employs critical pair analysis, a fundamental technique in rewriting theory. The idea is that  $t \cong u$  holds if replacing  $t$  with  $u$  (which means applying a refinement rule  $t \Rightarrow u$ ) in any program does not conflict with any evaluation rule  $l \rightarrow r$ .

## 1.1 Overview

We provide an overview of the new frameworks of *Term Evaluation Systems (TES)* and *Term Evaluation and Refinement Systems (TERS)* we formulate in this paper, using examples to illustrate their structure. We also demonstrate our main result: a method for deriving contextual improvement through local coherence.

The standard left-to-right call-by-value lambda-calculus is a TES. Terms  $t, t'$  including values  $v$  are defined as below, and the call-by-value evaluation strategy is specified using evaluation contexts  $E$  and one evaluation rule  $\rightarrow$ :

$$v ::= \lambda x.t, \quad t, t' ::= x \mid v \mid t \ t', \quad E ::= \square \mid E \ t \mid v \ E, \quad (\lambda x.t) \ v \rightarrow t[v/x].$$

Values  $v$  appearing in this specification play a significant role. The definition of evaluation contexts notably includes the clause  $v \ E$  where the left subterm  $v$  is restricted to values. This ensures the left-to-right evaluation of application  $t \ t'$ ; the right subterm  $t'$  can be evaluated only after the left subterm  $t$  has been evaluated to a value. Additionally, the redex  $(\lambda x.t) \ v$  restricts the right subterm  $v$  to values. This ensures the call-by-value evaluation of application.

A simplified computational lambda-calculus  $\lambda_{ml*}$  [7] is a TERS. Its terms are either values  $v, v'$  or computations  $p, p'$ , and its evaluation (which has been studied [1]) is specified using evaluation contexts  $E$  and two evaluation rules  $\rightarrow$ :

$$\begin{aligned} v, v' &::= x \mid \lambda x.p, & p, p' &::= \mathbf{return}(v) \mid \mathbf{let} \ x = p \ \mathbf{in} \ p' \mid v \ v', \\ E &::= \square \mid \mathbf{let} \ x = E \ \mathbf{in} \ p, & (\lambda x.p) \ v &\rightarrow p[v/x], \ \mathbf{let} \ x = \mathbf{return}(v) \ \mathbf{in} \ p \rightarrow p[v/x]. \end{aligned}$$

We can observe that evaluation contexts constrain where evaluation rules can be applied, namely in the subterm  $p$  of  $\mathbf{let} \ x = p \ \mathbf{in} \ p'$ . Again, values in evaluation rules assure the call-by-value evaluation of application and let-binding.

Originally, the calculus  $\lambda_{ml*}$  is specified by equations rather than evaluation. Directed

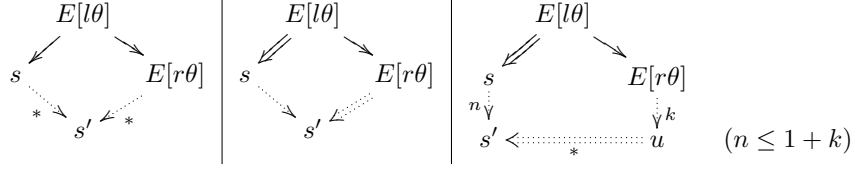


Figure 1: Joinability for confluence, commutation and local coherence

equations can be seen as the following five refinement rules  $\Rightarrow$ :

$$\begin{aligned}
 (\lambda x.p) v &\Rightarrow p[v/x], & \text{let } x = \mathbf{return}(v) \text{ in } p &\Rightarrow p[v/x], \\
 \lambda x.v \ x &\Rightarrow v, & \text{let } x = p \text{ in } \mathbf{return}(x) &\Rightarrow p, \\
 \text{let } x_1 = (\text{let } x_2 = p_2 \text{ in } p_1) \text{ in } p_3 &\Rightarrow \text{let } x_2 = p_2 \text{ in let } x_1 = p_1 \text{ in } p_3.
 \end{aligned}$$

While the first two rules represent  $\beta$ -conversion, the third one represents  $\eta$ -conversion. The fourth one removes the trivial let-binding, and the last one flattens let-bindings. We can observe that the last three rules *simplify* terms. We now have a TERS of  $\lambda_{ml*}$  which has both evaluation and refinement. We are now interested in whether refinement is *valid* with respect to evaluation. Our goal here is namely to prove contextual improvement: that is, for any refinement  $t \Rightarrow_{\mathcal{R}} u$  and any context  $C \in \text{Ctx}$ , if evaluation of  $C[t]$  terminates, then evaluation of  $C[u]$  terminates with no more evaluation steps. To prove contextual improvement, we would need to analyse how each evaluation step interferes with the refinement  $t \Rightarrow_{\mathcal{R}} u$ . This amounts to analyse how each evaluation rule  $l \rightarrow r$  can *conflict with* each refinement rule  $l' \Rightarrow r'$ . This is what exactly critical pair analysis is targeted at. Critical pair analysis is usually for proving confluence, which is a fundamental property of term rewriting. It firstly enumerates the situation where two rewrite rules conflict with each other. It then checks if the two conflicting rewritings can be *joined*. This is illustrated in Fig. 1 (left), where the joining part is depicted in dashed arrows, and ‘\*’ means an arbitrary number of rewriting. In our development, we exploit critical pair analysis for proving contextual improvement, and more specifically for proving **local coherence**. The analysis is targeted at conflicts between evaluation  $\rightarrow$  and refinement  $\Rightarrow$ . We analyse if these conflicts can be *joined* using evaluation and refinement; see Fig. 1 (right). To ensure improvement, our notion of local coherence asserts that the joining part satisfies the inequality  $1 + k \geq n$  about the number of evaluation steps. To prove the joinability for local coherence, we need to be careful with evaluation contexts. We need to show that the  $1 + k$  evaluation steps  $E[l\theta] \rightarrow_{\mathcal{E}} E[r\theta] \xrightarrow{k}_{\mathcal{E}} u$  can be *simulated* by the  $n$  evaluation steps  $s \xrightarrow{n}_{\mathcal{E}} s'$ . Naively, this can be done by showing that the evaluation rule  $l \rightarrow r$  can also be applied to the term  $s$ . This, however, involves making sure that the rule  $l \rightarrow r$  can be applied *inside an evaluation context*. This is not a trivial issue; the evaluation context  $E$  might be modified by the refinement  $E[t] \Rightarrow_{\mathcal{R}} s$ . This modification should be “mild”, and more precisely, refinement should not turn an evaluation context into a non-evaluation context. Note that local coherence can be seen as a generalisation of *commutation* [9]; see Fig. 1 (middle). Commutation is the case where  $k = 0$ ,  $n = 1$ , and allowing only one step of refinement  $\Rightarrow_{\mathcal{R}}$  instead of  $\xRightarrow{*}_{\mathcal{R}}$ . Our main theorems are as follows.

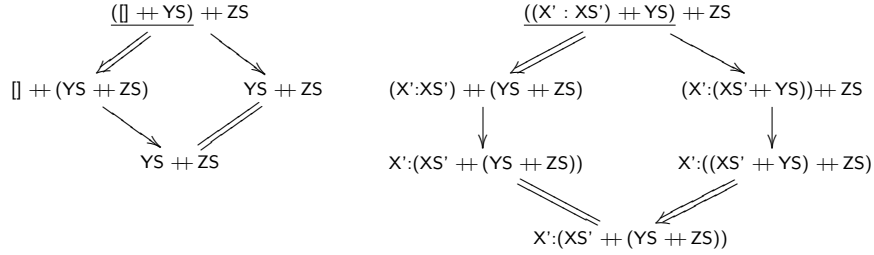
**Theorem 1.** A well-behaved TERS is locally coherent if and only if every critical pair is joinable.

**Theorem 2.** Let  $(\mathcal{E}, \mathcal{R}, Val)$  be a TERS. If  $\mathcal{E}$  is deterministic, and  $(\mathcal{E}, \mathcal{R}, Val)$  is value-invariant and locally coherent, then the set  $\mathcal{R}$  of refinement rules is improvement w.r.t. the set  $\mathcal{E}$  of evaluation rules.

**Example 1.** We define a TERS **Append** as follows.

<b>Signature</b> $\Sigma$	$[], (:), (++) : 2$
<b>Values</b> $Val$	$V ::= [] \mid V : V$
<b>Evaluation contexts</b> $Ectx$	$E ::= \square \mid E ++ t \mid E : t \mid V : E$
<b>Evaluation rules</b> $\mathcal{E}$	<b>Refinement rules</b> $\mathcal{R}$
$[] ++ ys \rightarrow ys$	$(xs ++ ys) ++ zs \Rightarrow xs ++ (ys ++ zs)$
$(x : xs) ++ ys \rightarrow x : (xs ++ ys)$	

This defines a well-known append function on strict lists with associativity as a refinement rule. Equations that naturally hold for lists can be considered as candidates for refinement rules. The TERS **Append** has the following two joinable critical pairs.



Because the TERS is well-behaved, it is locally coherent by Thm. 1. By Thm. 2, we conclude that the refinement rule expressing the associativity of append is *improvement* w.r.t. its evaluation.

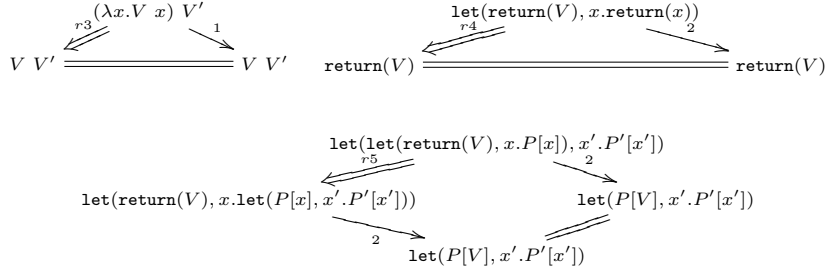
**Example 2.** (A simplified computational lambda-calculus  $\lambda_{ml*}$  [7, 1]) A notion of evaluation for Sabry and Wadler's computational lambda-calculus  $\lambda_{ml*}$  [7] has been studied [1]. A TERS **Comp** $\lambda_{ml*}$  of the computational lambda-calculus is defined as follows. We use syntactic sugar  $\lambda x.t \equiv \lambda(x.t), t u \equiv @(t, u)$ .

<b>Syntax class</b> $Sclass$	
<b>Values</b> $V, V' ::= x \mid \lambda x.P$	
<b>Computations</b> $P, P' ::= \text{return}(V) \mid \text{let}(P, x.P') \mid V V'$	
<b>Evaluation contexts</b> $Ectx$ $E ::= \square \mid \text{let}(E, x.P)$	
<b>Evaluation rules</b> $\mathcal{E}$	
$(\lambda x.P[x]) V \rightarrow P[V]$	(1)
$\text{let}(\text{return}(V), x.P[x]) \rightarrow P[V]$	(2)
<b>Refinement rules</b> $\mathcal{R}$	
$(\lambda x.P[x]) V \Rightarrow P[V]$	(r1)
$\text{let}(\text{return}(V), x.P[x]) \Rightarrow P[V]$	(r2)
$\lambda x.V x \Rightarrow V$	(r3)

$$\mathbf{let}(P, x.\mathbf{return}(x)) \Rightarrow P \quad (\text{r4})$$

$$\mathbf{let}(\mathbf{let}(P_1, x_1.P_2[x_1]), x_2.P_3[x_2]) \Rightarrow \mathbf{let}(P_1, x_1.\mathbf{let}(P_2[x_1], x_2.P_3[x_2])) \quad (\text{r5})$$

We define  $=_{val}$  by the total relation  $\top$ . This means we observe only termination (since  $Val \subseteq \mathbf{NF}(\rightarrow_{\mathcal{E}})$ ), identifying all values. The TERS  $\mathbf{Comp}\lambda_{ml*}$  has the following three critical pairs. In the following, arrows  $\rightarrow, \Rightarrow$  are labelled by a number that indicates which evaluation/refinement rule is applied.



Finally, by Thm. 1 and Thm. 2, the refinement rules  $\mathcal{R}$  of each of the TERSs are *improvement* with respect to the evaluation rules  $\mathcal{E}$ .

## References

- [1] Claudia Faggian, Giulio Guerrieri, and Riccardo Treglia. Evaluation in the computational calculus is non-confluent. In *10th International Workshop of Confluence, IWC 2021*, pages 31–36, 2021.
- [2] Matthias Felleisen. lambda-v-cs: An extended lambda-calculus for scheme. In Jérôme Chailloux, editor, *Proceedings of the 1988 ACM Conference on LISP and Functional Programming, LFP 1988, Snowbird, Utah, USA, July 25-27, 1988*, pages 72–85. ACM, 1988.
- [3] Matthias Felleisen. The theory and practice of first-class prompts. In Jeanne Ferrante and Peter Mager, editors, *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, California, USA, January 10-13, 1988*, pages 180–190. ACM Press, 1988.
- [4] James Hiram Morris Jr. *Lambda-calculus models of programming languages*. PhD thesis, Massachusetts Institute of Technology, 1969.
- [5] Koko Muroya. *Hypernet semantics of programming languages*. PhD thesis, University of Birmingham, 2020.
- [6] Koko Muroya and Makoto Hamana. Term evaluation systems with refinements: First-order, second-order, and contextual improvement. In Jeremy Gibbons and Dale Miller, editors, *Functional and Logic Programming - 17th International Symposium, FLOPS 2024, Kumamoto, Japan, May 15-17, 2024, Proceedings*, volume 14659 of *Lecture Notes in Computer Science*, pages 31–61. Springer, 2024.
- [7] Amr Sabry and Philip Wadler. A reflection on call-by-value. In Robert Harper and Richard L. Wexelblat, editors, *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming, ICFP 1996, Philadelphia, Pennsylvania, USA, May 24-26, 1996*, pages 13–24. ACM, 1996.
- [8] David Sands. Total correctness by local improvement in the transformation of functional programs. *ACM Trans. Program. Lang. Syst.*, 18(2):175–234, 1996.
- [9] Yoshihito Toyama. *Commutativity of term rewriting systems*, pages 393–407. North-Holland, 1988.

# Confluence of 001- and 101-infinitary $\lambda$ -calculi by linear approximation\*

R  my Cerda<sup>1,2,†</sup> and Lionel Vaux Auclair<sup>1</sup>

<sup>1</sup> Aix-Marseille Universit  , CNRS, I2M, France

<sup>2</sup> Universit   Paris Cit  , CNRS, IRIF, F-75013 Paris, France  
{Remy.Cerda,Lionel.Vaux}@math.cnrs.fr

The introduction of infinitary rewriting [DKP91; Ken+95] and in particular of infinitary  $\lambda$ -calculi [Ken+97] created a syntactic bridge between the *dynamics* of rewriting systems ( $\beta$ -reduction in the case of the  $\lambda$ -calculus, whose presentation is finitary while inducing infinite behaviours) and their *semantics* (at least in its most syntactic flavours, *e.g.* B  hm trees for the  $\lambda$ -calculus). Confluence, already a highly desirable property for a finitary rewriting system, becomes even more important in this setting as it ensures uniqueness of infinitary normal forms, *i.e.* consistency of the associated model. However, it is a fragile property as the infinitary closure of a confluent reduction may not be confluent (in the  $\lambda$ -calculus,  $\mathbf{YI} \rightarrow_{\beta}^{\infty} \mathbf{I}^{\omega} := \mathbf{III} \dots$  and  $\mathbf{YI} \rightarrow_{\beta}^{\infty} \Omega$ , which constitutes a critical pair), hence the need for tweaking this infinitary closure to retrieve confluence [KOV96; SV11b].

A seemingly orthogonal line of work originating in Girard’s quantitative semantics [Gir88] led to the advent of a linear approximation of the  $\lambda$ -calculus based on Taylor expansion [ER08; ER06], which allowed for a renewal and a refinement of the classic approach based on continuous approximation, and for a whole range of new, elegant proofs of key results in  $\lambda$ -calculus [BM20; CV23]. The major property of the linear approximation, known as the Commutation theorem, relates the infinitary head normalisation of a  $\lambda$ -term towards its B  hm tree to the (finitary) normalisation of its Taylor expansion, that is, the sum of its multilinear “resource” approximants.

However, this presentation of the linear approximation is slightly disappointing for at least two reasons: it only accounts for infinitary normalisation of  $\lambda$ -terms, instead of arbitrary  $\beta$ -reduction sequences; it relies on the continuous approximation to handle the Taylor expansion of B  hm trees, instead of being built independently. In [CV23; Cer24], we demonstrate how extending the linear approximation to an infinitary  $\lambda$ -calculus and relaxing Commutation (wrt. normalisation) into a property of Simulation (wrt. infinitary  $\beta\bot$ -reduction) allows to overcome these two impediments. Infinitary  $\lambda$ -calculus thus appears to be the “missing ingredient” thanks to which we could give a general, canonical presentation of the linear approximation of the  $\lambda$ -calculus.

In the following exposition, we take a somehow dual perspective and explain what linear approximation brings to infinitary  $\lambda$ -calculi. In section 1, we recall the coinductive presentation of *abc*-infinitary  $\lambda$ -calculi. In section 2, we present two linear approximations for the 001-infinitary  $\lambda$ -calculus (this was published in [CV23] and directly extends Ehrhard and Regnier’s original work), and for the 101-infinitary  $\lambda$ -calculus (which is not yet formally published, but has already been presented in [Cer24; Cer]). In both cases, we state a Simulation theorem relating the infinitary  $\beta\bot$ -reduction to the reduction of Taylor expansions. In section 3, we detail how confluence of the given infinitary  $\lambda$ -calculi can be deduced as a corollary of Simulation. Finally, in section 4 we evoke the remaining infinitary  $\lambda$ -calculi (the 111-infinitary version, which is also confluent, and more generally the infinitary  $\lambda$ -calculi modulo meaningless terms), stating a

\*Abstract submitted to the 14th International Workshop on Confluence (IWC 2025).

†Partially funded by the ANR project RECIPROG ANR-21-CE48-019.

negative result preventing the construction of a suitable linear approximation. We also mention possible directions for further work.

## 1 Infinitary $\lambda$ -calculi

We first recall the construction of infinitary  $\lambda$ -calculi. We depart from the original definition [Ken+97] and follow its coinductive reformulation [Joa04; EP13; Cer24].

Fix a countable set  $\mathcal{V}$  of variables. Recall the inductive syntax of finite  $\lambda$ -terms:

$$\frac{x \in \mathcal{V}}{x \in \Lambda} \quad \frac{x \in \mathcal{V} \quad P \in \Lambda}{\lambda x.P \in \Lambda} \quad \frac{P \in \Lambda \quad Q \in \Lambda}{PQ \in \Lambda}$$

By treating these rules coinductively, one obtains a set of infinitary  $\lambda$ -terms. But one could also treat each constructor in a different way (inductive or coinductive), which is the point of the following definition. Take  $a, b, c \in \{0, 1\}$ , then the set  $\Lambda^{abc}$  of  $abc$ -infinitary  $\lambda$ -terms is defined by:

$$\frac{x \in \mathcal{V}}{x \in \Lambda^{abc}} \quad \frac{x \in \mathcal{V} \quad \triangleright_a P \in \Lambda^{abc}}{\lambda x.P \in \Lambda^{abc}} \quad \frac{\triangleright_b P \in \Lambda^{abc} \quad \triangleright_c Q \in \Lambda^{abc}}{PQ \in \Lambda^{abc}} \quad \frac{M \in \Lambda^{abc}}{\triangleright_0 M \in \Lambda^{abc}} \quad \frac{M \in \Lambda^{abc}}{\triangleright_1 M \in \Lambda^{abc}}$$

where only the last rule is coinductive, *i.e.* infinite branches in infinite derivations must cross this rule (and hence the coinductive guard  $\triangleright_1$ ) infinitely often. In particular,  $\Lambda^{000} = \Lambda$ .

These sets are implicitly quotiented by  $\alpha$ -equivalence, which is very standard for finite  $\lambda$ -terms but raises certain technicalities for infinitary ones; a complete treatment using nominal sets is provided in [Kur+13; Cer25]. This allows to define capture-avoiding substitution at the level of  $\alpha$ -equivalence classes, and we denote by  $M[N/x]$  the term obtained by substituting  $N$  to all free occurrences of  $x$  in  $M$ .

All our sets of  $\lambda$ -terms are endowed with the relation  $\longrightarrow_\beta$  of  $\beta$ -reduction, defined by  $(\lambda x.M)N \longrightarrow_\beta M[N/x]$  and by inductively lifting to contexts. Given again  $a, b, c \in \{0, 1\}$ , the  $abc$ -infinitary closure of  $\beta$ -reduction is defined by:

$$\frac{M \longrightarrow_\beta^* x}{M \longrightarrow_\beta^{abc} x} \quad \frac{M \longrightarrow_\beta^* \lambda x.P \quad \triangleright_a P \longrightarrow_\beta^{abc} P'}{M \longrightarrow_\beta^{abc} \lambda x.P'} \quad \frac{M \longrightarrow_\beta^* PQ \quad \triangleright_b P \longrightarrow_\beta^{abc} P' \quad \triangleright_c Q \longrightarrow_\beta^{abc} Q'}{M \longrightarrow_\beta^{abc} P'Q'} \quad \frac{M \longrightarrow_\beta^{abc} N}{\triangleright_0 M \longrightarrow_\beta^{abc} N} \quad \frac{M \longrightarrow_\beta^{abc} N}{\triangleright_1 M \longrightarrow_\beta^{abc} N}$$

We use the standard combinators  $I := \lambda x.x$ ,  $K := \lambda xy.x$ ,  $\Omega := (\lambda x.xx)(\lambda x.xx)$ , and the fixed-point combinator  $Y := \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$  (such that  $Yf =_\beta f(Yf)$ ). We can define the following examples of infinitary  $\lambda$ -terms:

$$M^\omega := M(M(M(\dots))) \in \Lambda^{ab1} \text{ for } M \in \Lambda^{ab1} \quad O := \lambda y_0.\lambda y_1.\lambda y_2.\dots \in \Lambda^{1bc}$$

as well as the infinitary  $\beta$ -reductions:

$$Yf \longrightarrow_\beta^{ab1} f^\omega \quad YK \longrightarrow_\beta^{1bc} O$$

As said, the  $abc$ -infinitary closures lack confluence (unless  $abc = 000$ ), which is a critical issue. We already mentioned the critical pair  $YI \longrightarrow_\beta^{ab1} I^\omega$  and  $YI \longrightarrow_\beta \Omega$ , let us consider the following variant that we will use as a running example:

$$YK \longrightarrow_\beta^{001} K^\omega \quad YK \longrightarrow_\beta^* (\lambda xy.xx)(\lambda xy.xx). \quad (1)$$

The standard solution to restore confluence is to extend  $\beta$ -reduction with a relation  $\longrightarrow_{\perp}$  of  $\perp$ -reduction collapsing the “problematic” parts of a term to a constant  $\perp$ . We denote by  $\Lambda_{\perp}$ ,  $\Lambda_{\perp}^{abc}$  the sets obtained by the above definitions with additional axiom rules saying that  $\perp \in \Lambda_{\perp}$  and  $\perp \in \Lambda_{\perp}^{abc}$ . Given a set  $\mathcal{U} \subseteq \Lambda_{\perp}^{abc}$ , a reduction  $\longrightarrow_{\perp \mathcal{U}}$  is defined on  $\Lambda_{\perp}^{abc}$  by  $M \longrightarrow_{\perp \mathcal{U}} \perp$  for all  $M \in \mathcal{U}$ , and by inductively lifting to contexts. We also define  $\longrightarrow_{\beta \perp \mathcal{U}} := \longrightarrow_{\beta} \cup \longrightarrow_{\perp \mathcal{U}}$ .

Remember that a  $\lambda$ -term is either a *head normal form* (HNF), *i.e.* a term of the shape  $\lambda x_1 \dots \lambda x_m. y M_1 \dots M_n$ , or a term  $\lambda x_1 \dots \lambda x_m. (\lambda x. P) Q M_1 \dots M_n$  where  $(\lambda x. P) Q$  is called the head redex. This can be refined as follows: a  $\lambda$ -term is either a term  $\lambda x. M$ , or a term  $y M_1 \dots M_n$  (two types of *weak head normal forms*, or WHNF’s), or a term  $(\lambda x. P) Q M_1 \dots M_n$  where  $(\lambda x. P) Q$  is called the weak head redex.

For  $\mathcal{U}$  in the definitions above, we may in particular consider the following sets:

$$\overline{\mathcal{HN}} := \{M \in \Lambda_{\perp}^{111} \mid M \text{ has no HNF}\} \quad \overline{\mathcal{WHN}} := \{M \in \Lambda_{\perp}^{111} \mid M \text{ has no WHNF}\}.$$

We define  $\longrightarrow_{\beta \perp}^{001}$  (resp.  $\longrightarrow_{\beta \perp}^{101}$ ) on  $\Lambda_{\perp}^{001}$  (resp.  $\Lambda_{\perp}^{101}$ ) by the rules defining  $\longrightarrow_{\beta}^{001}$  (resp.  $\longrightarrow_{\beta}^{101}$ ), where we replace  $\longrightarrow_{\beta}^*$  with  $\longrightarrow_{\beta \perp \overline{\mathcal{HN}}}^*$  (resp.  $\longrightarrow_{\beta \perp \overline{\mathcal{WHN}}}^*$ ).

Standard examples are given by the coinductive definitions of the Böhm tree of a term  $M \in \Lambda_{\perp}^{001}$ :

$$\text{BT}(M) := \begin{cases} \lambda x_1 \dots \lambda x_m. y \text{BT}(M_1) \dots \text{BT}(M_n) & \text{if } M \longrightarrow_h^* \lambda x_1 \dots \lambda x_m. y M_1 \dots M_n, \\ \perp & \text{otherwise,} \end{cases}$$

where  $\longrightarrow_h$  denotes head reduction, *i.e.* the restriction of  $\beta$ -reduction where one only reduces head redexes, and of the Lévy-Longo tree of a term  $M \in \Lambda_{\perp}^{101}$ :

$$\text{LLT}(M) := \begin{cases} \lambda x. \text{LLT}(M') & \text{if } M \longrightarrow_{\text{wh}}^* \lambda x. M' \\ y \text{LLT}(M_1) \dots \text{LLT}(M_n) & \text{if } M \longrightarrow_{\text{wh}}^* y M_1 \dots M_n, \\ \perp & \text{otherwise,} \end{cases}$$

where  $\longrightarrow_{\text{wh}}$  denotes weak head reduction, *i.e.* the restriction of  $\beta$ -reduction where one only reduces weak head redexes. It is easy to verify that  $M \longrightarrow_{\beta \perp}^{001} \text{BT}(M)$  and  $M \longrightarrow_{\beta \perp}^{101} \text{LLT}(M)$ , just following from their definition. In particular, for the previously introduced examples:

$$\text{BT}(Yf) = \text{LLT}(Yf) = f^{\omega} \quad \text{BT}(YI) = \text{LLT}(YI) = \perp \quad \text{BT}(YK) = \perp \quad \text{LLT}(YK) = \mathbf{0}.$$

## 2 Strict and lazy linear approximations

The linear approximation relies on a map  $\mathcal{T} : \Lambda_{\perp}^{001} \rightarrow \mathcal{P}(\Lambda_r)$  mapping  $\lambda$ -terms to sets<sup>1</sup> of “resource  $\lambda$ -terms”, the terms of a multilinear  $\lambda$ -calculus. We recall its construction very briefly, see [VA19; Cer24] for more details. The set  $\Lambda_r$  of resource  $\lambda$ -terms is defined inductively by:

$$\begin{aligned} \Lambda_r &\ni s, t, \dots &:= x \mid \lambda x. s \mid s\bar{t} && (x \in \mathcal{V}) \\ !\Lambda_r &\ni \bar{t}, \bar{u}, \dots &:= [t_1, \dots, t_n] && (n \in \mathbf{N}) \end{aligned}$$

We write  $(!) \Lambda_r$  to denote  $\Lambda_r$  or  $!\Lambda_r$ . We denote by  $\mathbf{N}[(!) \Lambda_r]$  the  $\mathbf{N}$ -semimodule of finitely supported formal sums of resource  $\lambda$ -terms (finite resource sums in short). We denote by boldface  $\mathbf{s}, \mathbf{t}$ , etc. its elements and by  $\mathbf{0}$  the empty sum. As usual in resource  $\lambda$ -calculus, we

<sup>1</sup>In the general, quantitative definition, the Taylor expansion is an infinite formal sum of resource  $\lambda$ -terms weighted by coefficients taken in an arbitrary semiring with fractions. Here we present the qualitative definition where we just work with the semiring of booleans, thus the formal sums are mere sets.

assimilate resource terms to one-element resource sums and we extend the constructors of the above inductive definitions to sums, by linearity (*e.g.*,  $\lambda x.(s + \mathbf{t}) = \lambda x.s + \lambda x.\mathbf{t}$  or  $\mathbf{0}\bar{t} = \mathbf{0}$ ).

Substitution in resource terms is defined by

$$s\langle [t_1, \dots, t_n]/x \rangle := \begin{cases} \sum_{\sigma \in \mathfrak{S}(n)} s[t_{\sigma(1)}/x_1, \dots, t_{\sigma(n)}/x_n] & \text{if } \deg_x(s) = n \\ \mathbf{0} & \text{otherwise,} \end{cases}$$

where  $\deg_x(s)$  is the number of free occurrences of  $x$  in  $s$ ,  $x_1, \dots, x_n$  is an arbitrary enumeration of these free occurrences,  $\mathfrak{S}(n)$  is the set of all permutations of  $\{1, \dots, n\}$ , and  $s[t_{\sigma(1)}/x_1, \dots, t_{\sigma(n)}/x_n]$  is the resource term obtained by substituting the  $t_i$  to the occurrences  $x_i$ .

The relation  $\longrightarrow_r$  of resource reduction is defined as a subset of  $(!) \Lambda_r \times \mathbf{N}[(!) \Lambda_r]$  by  $(\lambda x.s)\bar{t} \longrightarrow_r s\langle \bar{t}/x \rangle$  and by lifting to contexts. It is extended to a relation on  $\mathbf{N}[\Lambda_r]$  by saying that  $s + \mathbf{t} \longrightarrow_r s' + \mathbf{t}'$  whenever  $s \longrightarrow_r s'$  and  $\mathbf{t} \longrightarrow_r^? \mathbf{t}'$  ( $\longrightarrow_r^?$  denoting the reflexive closure).

**Lemma 1.**  $\longrightarrow_r$  is confluent<sup>2</sup> and strongly normalising.

Let us now define the linear approximation itself. A relation  $\sqsubseteq_{\mathcal{T}}$  is defined as a subset of  $\Lambda_r \times \Lambda_{\perp}^{001}$  by the following inductive rules:

$$\frac{}{x \sqsubseteq_{\mathcal{T}} x} \quad \frac{s \sqsubseteq_{\mathcal{T}} M}{\lambda x.s \sqsubseteq_{\mathcal{T}} \lambda x.M} \quad \frac{s \sqsubseteq_{\mathcal{T}} M \quad t_1 \sqsubseteq_{\mathcal{T}} N \quad \dots \quad t_n \sqsubseteq_{\mathcal{T}} N}{s[t_1, \dots, t_n] \sqsubseteq_{\mathcal{T}} (M)N}$$

and the Taylor expansion of any  $M \in \Lambda_{\perp}^{001}$  is defined by  $\mathcal{T}(M) := \{s \in \Lambda_r \mid s \sqsubseteq_{\mathcal{T}} M\}$ . For example,  $\mathcal{T}(f^\omega)$  contains  $f[], f[f[]], f[f[], f[]], f[f[f[]]]$ , and can be described more generally by the following inductive equation:  $\mathcal{T}(f^\omega) = \{f[t_1, \dots, t_n] \mid n \in \mathbf{N}, t_1, \dots, t_n \in \mathcal{T}(f^\omega)\}$ .

Since Taylor expansion maps  $\lambda$ -terms to sets of resource terms, we need to explain how to lift the resource reduction to such sets. Let us denote by  $|s|$  the support of any finite sum  $s \in \mathbf{N}[(!) \Lambda_r]$ . Then for all  $S, T \subseteq (!) \Lambda_r$  we write  $S \longrightarrow_r T$  whenever there is a set  $I$  such that  $S = \bigcup_{i \in I} \{s_i\}$ ,  $T = \bigcup_{i \in I} |t_i|$  and for all  $i \in I$ ,  $s_i \longrightarrow_r^* t_i$ .

Thanks to lemma 1, we can also define  $\text{nf}_r(s)$  to be the unique normal form through  $\longrightarrow_r$  of any  $s \in \mathbf{N}[(!) \Lambda_r]$ , and  $\text{nf}_r(S) := \bigcup_{s \in S} |\text{nf}_r(s)|$  for all set  $S \subseteq (!) \Lambda_r$ . In particular,  $S \longrightarrow_r \text{nf}_r(S)$ .

Our main result in [CV23] is the following theorem, that can be seen as the cornerstone of the whole linear approximation theory for the  $\lambda$ -calculus. An immediate consequence is Ehrhard and Regnier's celebrated Commutation theorem.

**Theorem 2** (Simulation). For all  $M, N \in \Lambda_{\perp}^{001}$ , if  $M \longrightarrow_{\beta\perp}^{001} N$  then  $\mathcal{T}(M) \longrightarrow_r \mathcal{T}(N)$ .

**Corollary 3** (Commutation). For all  $M \in \Lambda_{\perp}^{001}$ ,  $\text{nf}_r(\mathcal{T}(M)) = \mathcal{T}(\text{BT}(M))$ .

As noticed in [Cer24; Cer] one can adapt this work to the lazy setting, *i.e.* to weak head normalisation and the 101-infinitary  $\lambda$ -calculus. To do so, we add a constant  $\mathbf{o}$  in the syntax of the resource  $\lambda$ -calculus (it stands for an undefined abstraction, typically approximating of  $\lambda x.\perp$ ), defining a set of lazy resource  $\lambda$ -terms  $\Lambda_{r\ell}$ . The lazy Taylor expansion of any  $M \in \Lambda_{\perp}^{101}$  is defined by  $\mathcal{T}_{\ell}(M) := \{s \in \Lambda_{r\ell} \mid s \sqsubseteq_{\mathcal{T}_{\ell}} M\}$ , where  $\sqsubseteq_{\mathcal{T}_{\ell}}$  is defined just as  $\sqsubseteq_{\mathcal{T}}$  with an additional rule saying that  $\mathbf{o} \sqsubseteq_{\mathcal{T}_{\ell}} \lambda x.M$  for all  $M$ . We obtain similar results:

**Theorem 4** (Simulation). For all  $M, N \in \Lambda_{\perp}^{101}$ , if  $M \longrightarrow_{\beta\perp}^{101} N$  then  $\mathcal{T}_{\ell}(M) \longrightarrow_r \mathcal{T}_{\ell}(N)$ .

**Corollary 5** (Commutation). For all  $M \in \Lambda_{\perp}^{101}$ ,  $\text{nf}_r(\mathcal{T}_{\ell}(M)) = \mathcal{T}_{\ell}(\text{LLT}(M))$ .

<sup>2</sup>In fact a way stronger property holds: its reflexive closure  $\longrightarrow_r^?$  has the diamond property.



### 3 Confluence results

Before we show how the linear approximation allows for elementary proofs of confluence for  $\rightarrow_{\beta\perp}^{001}$  and  $\rightarrow_{\beta\perp}^{101}$ , let us describe how confluence works on the example from eq. (1), namely

$$\Upsilon K \rightarrow_{\beta}^{a01} K^{\omega} \quad \Upsilon K \rightarrow_{\beta}^* (\lambda xy.xx)(\lambda xy.xx),$$

a critical pair for  $\rightarrow_{\beta\perp}^{001}$ , but neither for  $\rightarrow_{\beta\perp}^{001}$  nor for  $\rightarrow_{\beta\perp}^{101}$ , for two different reasons.

In the latter case, the reductions can simply be continued:

$$K^{\omega} = KK^{\omega} \rightarrow_{\beta} \lambda y.K^{\omega} \rightarrow_{\beta}^{101} O \quad (\lambda xy.xx)(\lambda xy.xx) \rightarrow_{\beta} \lambda y.(\lambda xy.xx)(\lambda xy.xx) \rightarrow_{\beta}^{101} O.$$

In the former case however, this cannot be done as it is forbidden to reduce coinductively under abstractions. Instead,  $\rightarrow_{\perp\overline{HN}}$  (included in  $\rightarrow_{\beta\perp}^{001}$ ) restores confluence:

$$K^{\omega} \rightarrow_h \lambda y.K^{\omega} \rightarrow_{\perp\overline{HN}} \perp \quad (\lambda xy.xx)(\lambda xy.xx) \rightarrow_h \lambda y.(\lambda xy.xx)(\lambda xy.xx) \rightarrow_{\perp\overline{HN}} \perp.$$

(From the first head reduction steps we explicitly wrote, one can see that no HNF will be reached.) Through the lens of Taylor expansion, the first reduction (and similarly the second) can be seen as  $\mathcal{T}(K^{\omega}) \rightarrow_r \emptyset$ . Indeed, recall from a previous observation that  $\mathcal{T}(K^{\omega})$  can be described by the equation  $\mathcal{T}(K^{\omega}) = \{K[t_1, \dots, t_n] \mid n \in \mathbf{N}, t_1, \dots, t_n \in \mathcal{T}(K^{\omega})\}$ . Since this inductive construction has  $K[]$  as its base case, any  $s \in \mathcal{T}(K^{\omega})$  contains  $K[]$  as a subterm, and since  $K[] \rightarrow_r O$  any such  $s$  also collapses to  $O$  by linearity.

This example shows how the collapse of any “non-001” behaviour, like the production of  $O$ , is erased by the resource reduction in the world of Taylor expansions. This is hidden in the following proof of theorem 8.

**Lemma 6.** If  $N \in \Lambda_{\perp}^{001}$  is in normal form for  $\rightarrow_{\beta\perp\overline{HN}}$ , then  $\text{BT}(N) = N$ .

**Lemma 7.**  $\mathcal{T}$  is injective when restricted to terms of  $\Lambda_{\perp}^{001}$  not containing subterms of the shape  $\lambda x.\perp$  or  $(\perp)M$ . In particular, it is injective on normal forms for  $\rightarrow_{\perp\overline{HN}}$ .

**Theorem 8** (uniqueness of normal forms). For all  $M \in \Lambda_{\perp}^{001}$ ,  $\text{BT}(M)$  is the unique normal form for  $\rightarrow_{\beta\perp\overline{HN}}$  reachable through  $\rightarrow_{\beta\perp}^{001}$  from  $M$ <sup>3</sup>.

*Proof.* Suppose there is another such normal form, denote it by  $N$ . Then:

$$\begin{aligned} \mathcal{T}(N) &= \mathcal{T}(\text{BT}(N)) && \text{by lemma 6,} \\ &= \text{nf}_r(\mathcal{T}(N)) && \text{by corollary 3,} \\ &= \text{nf}_r(\mathcal{T}(M)) && \text{by theorem 2 and lemma 1,} \\ &= \mathcal{T}(\text{BT}(M)) && \text{by corollary 3 again,} \end{aligned}$$

and we can conclude that  $N = \text{BT}(M)$  by lemma 7.  $\square$

**Corollary 9** (confluence).  $\rightarrow_{\beta\perp}^{001}$  is confluent on  $\Lambda_{\perp}^{001}$ .

*Proof.* If  $M \rightarrow_{\beta\perp}^{001} N$  and  $M \rightarrow_{\beta\perp}^{001} N'$ , then  $M \rightarrow_{\beta\perp}^{001} \text{BT}(N)$  and  $M \rightarrow_{\beta\perp}^{001} \text{BT}(N')$ , and  $\text{BT}(N) = \text{BT}(N') = \text{BT}(M)$  by theorem 8.  $\square$

The very same work can be done starting from theorem 4, giving rise to:

**Theorem 10** (uniqueness of normal forms). For all  $M \in \Lambda_{\perp}^{101}$ ,  $\text{LLT}(M)$  is the unique normal form for  $\rightarrow_{\beta\perp\overline{WN}}$  reachable through  $\rightarrow_{\beta\perp}^{101}$  from  $M$ .

**Corollary 11** (confluence).  $\rightarrow_{\beta\perp}^{101}$  is confluent on  $\Lambda_{\perp}^{101}$ .

<sup>3</sup>The reason why we do not simply write “the unique normal form for  $\rightarrow_{\beta\perp}^{001}$ ” is just that the latter relation is reflexive, hence not having any normal forms.

## 4 Beyond 001 and 101: a negative result and further work

In this section, we work in  $\Lambda_{\perp}^{111}$ , and we simply denote this set by  $\Lambda_{\perp}^{\infty}$ . In section 1, we defined a reduction  $\rightarrow_{\perp\mathcal{U}}$  collapsing any set  $\mathcal{U} \subseteq \Lambda_{\perp}^{\infty}$  to  $\perp$ , but only used it for the two subsets  $\overline{\mathcal{HN}}$  and  $\overline{\mathcal{WN}}$ . This can in fact be seen as a general construction for restoring confluence of the infinitary  $\beta$ -reduction, relying on a notion of “meaningless set” defined by a certain list of axioms (see [KOV96; SV11b]), such that in particular  $\overline{\mathcal{HN}}$  and  $\overline{\mathcal{WN}}$  are meaningless sets.

**Theorem 12** ([KOV96; SV11b]). For all meaningless set  $\mathcal{U} \subseteq \Lambda_{\perp}^{\infty}$ , the reduction  $\rightarrow_{\beta\perp\mathcal{U}}^{\infty}$  is confluent. In addition, each  $M \in \Lambda_{\perp}^{\infty}$  has a unique normal form through  $\rightarrow_{\beta\perp\mathcal{U}}^{\infty}$ .

Notice that the instance of this theorem for  $\mathcal{U} := \overline{\mathcal{HN}}$  (resp.  $\mathcal{U} := \overline{\mathcal{WN}}$ ) is not exactly corollary 9 (resp. corollary 11), since we now work in  $\Lambda_{\perp}^{111}$ . However, the former can be easily deduced from the latter.

If we denote by  $T_{\mathcal{U}}(-)$  the map taking  $\lambda$ -terms to their normal form through  $\rightarrow_{\beta\perp\mathcal{U}}^{\infty}$  (so that in particular  $T_{\overline{\mathcal{HN}}} = \text{BT}$  and  $T_{\overline{\mathcal{WN}}} = \text{LLT}$ ), the equivalence relation generated by equating  $M$  and  $N$  whenever  $T_M = T_N$  induces a  $\lambda$ -model, called “normal form model”. These models form a lattice of cardinality  $2^c$ , where  $c$  is the cardinality of the continuum [SV11a]. By exploiting the semantic properties of these models, Severi and de Vries were able to distinguish BT and LLT from all other normal form models:

**Theorem 13** ([SV05a]).  $\overline{\mathcal{HN}}$  and  $\overline{\mathcal{WN}}$  are the only meaningless sets  $\mathcal{U}$  such that  $T_{\mathcal{U}} : \Lambda_{\perp}^{\infty} \rightarrow \Lambda_{\perp}^{\infty}$  is Scott-continuous (with respect to the standard approximation order on  $\Lambda_{\perp}^{\infty}$ ).

Notice that the approximation order on  $\lambda$ -terms corresponds to inclusion of Taylor expansions (in both the strict and the lazy setting), which means that Taylor expansion is Scott-continuous. As a consequence of this observation and Commutation (corollaries 3 and 5), one obtains the content of theorem 13 for  $\overline{\mathcal{HN}}$  and  $\overline{\mathcal{WN}}$ , *i.e.* that BT and LLT are continuous maps.

The fact that this is a straightforward consequence of the two main properties of the Taylor approximations gives another meaning to theorem 13: for all meaningless set  $\mathcal{U}$  different from  $\overline{\mathcal{HN}}$  and  $\overline{\mathcal{WN}}$ , there cannot be a Taylor expansion enjoying desirable properties, as a commutation theorem with respect to  $T_{\mathcal{U}}$ . In particular the other standard notion of infinite normal form for  $\lambda$ -terms, namely Berarducci trees [Ber96], does not enjoy such a Taylor expansion.

A possible workaround would be to consider another ordering on  $\Lambda_{\perp}^{\infty}$ , as introduced in [SV05b], which makes  $T_{\mathcal{U}}$  monotonous as soon as  $\mathcal{U}$  is “quasi-regular” (which is the case in particular for Berarducci trees): one could wonder whether a linear approximation compatible with such an ordering can be constructed.

Another avenue for future work is to investigate linear approximations in more complex rewriting systems which may not have been studied from the perspective of infinitary rewriting theory, but do enjoy notions of infinitary normal forms, *e.g.* probabilistic  $\lambda$ -calculi [DL19] or the  $\Lambda\mu$ -calculus [Sau10].

## References

- [Ber96] Alessandro Berarducci. “Infinite  $\lambda$ -calculus and non-sensible models”. In: *Logic and Algebra*. Routledge, 1996, pp. 339–377. DOI: [10.1201/9780203748671-17](https://doi.org/10.1201/9780203748671-17).
- [BM20] Davide Barbarossa and Giulio Manzonetto. “Taylor Subsumes Scott, Berry, Kahn and Plotkin”. In: *47th Symp. on Princ. of Prog. Lang.* 1. Proc. ACM Prog. Lang., 2020. DOI: [10.1145/3371069](https://doi.org/10.1145/3371069).

- [Cer] Rémy Cerde. “The lazy evaluation of the  $\lambda$ -calculus enjoys linear approximation, and that’s all”. Unpublished note. An abstract has appeared in the proceedings of TLLA 2025, see <https://lipn.univ-paris13.fr/TLLA/2025/abstracts/18-cerde.pdf>.
- [Cer24] Rémy Cerde. “Taylor Approximation and Infinitary  $\lambda$ -Calculi”. Theses. Aix-Marseille Université, 2024. URL: <https://hal.science/tel-04664728>.
- [Cer25] Rémy Cerde. *Nominal Algebraic-Coalgebraic Data Types, with Applications to Infinitary  $\lambda$ -Calculi*. To appear in the proceedings of FICS 2024. 2025. URL: <https://www.i2m.univ-amu.fr/perso/remy.cerde/fichiers/papiers/nominal-nu-mu-fics.pdf>.
- [CV23] Rémy Cerde and Lionel Vaux Auclair. “Finitary Simulation of Infinitary  $\beta$ -Reduction via Taylor Expansion, and Applications”. In: *Logical Methods in Computer Science* 19 (4 2023). DOI: [10.46298/LMCS-19\(4:34\)2023](https://doi.org/10.46298/LMCS-19(4:34)2023).
- [DKP91] Nachum Dershowitz, Stéphane Kaplan, and David A. Plaisted. “Rewrite, rewrite, rewrite, rewrite, ...”. In: *Theoretical Computer Science* 83.1 (1991), pp. 71–96. DOI: [10.1016/0304-3975\(91\)90040-9](https://doi.org/10.1016/0304-3975(91)90040-9).
- [DL19] Ugo Dal Lago and Thomas Leventis. “On the Taylor Expansion of Probabilistic Lambda Terms”. In: *4th International Conference on Fromal Structures for Computation and Deduction*. Ed. by Herman Geuvers. LIPIcs 131. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. DOI: [10.4230/LIPIcs.FSCD.2019.13](https://doi.org/10.4230/LIPIcs.FSCD.2019.13).
- [EP13] Jörg Endrullis and Andrew Polonsky. “Infinitary Rewriting Coinductively”. In: *TYPES 2011*. 2013, pp. 16–27. DOI: [10.4230/LIPIcs.TYPES.2011.16](https://doi.org/10.4230/LIPIcs.TYPES.2011.16).
- [ER06] Thomas Ehrhard and Laurent Regnier. “Böhm Trees, Krivine’s Machine and the Taylor Expansion of Lambda-Terms”. In: *Logical Approaches to Computational Barriers*. Ed. by Arnold Beckmann et al. Berlin, Heidelberg: Springer, 2006, pp. 186–197. DOI: [10.1007/11780342\\_20](https://doi.org/10.1007/11780342_20).
- [ER08] Thomas Ehrhard and Laurent Regnier. “Uniformity and the Taylor expansion of ordinary lambda-terms”. In: *Theoretical Computer Science* 403.2 (2008), pp. 347–372. DOI: [10.1016/j.tcs.2008.06.001](https://doi.org/10.1016/j.tcs.2008.06.001).
- [Gir88] Jean-Yves Girard. “Normal functors, power series and  $\lambda$ -calculus”. In: *Annals of Pure and Applied Logic* 37.2 (1988), pp. 129–177. DOI: [10.1016/0168-0072\(88\)90025-5](https://doi.org/10.1016/0168-0072(88)90025-5).
- [Joa04] Felix Joachimski. “Confluence of the coinductive  $\lambda$ -calculus”. In: *Theoretical Computer Science* 311.1-3 (2004), pp. 105–119. DOI: [10.1016/S0304-3975\(03\)00324-4](https://doi.org/10.1016/S0304-3975(03)00324-4).
- [Ken+95] Richard Kennaway et al. “Transfinite Reductions in Orthogonal Term Rewriting Systems”. In: *Information and Computation* 119.1 (1995), pp. 18–38. DOI: [10.1006/inco.1995.1075](https://doi.org/10.1006/inco.1995.1075).
- [Ken+97] Richard Kennaway et al. “Infinitary lambda calculus”. In: *Theoretical Computer Science* 175.1 (1997), pp. 93–125. DOI: [10.1016/S0304-3975\(96\)00171-5](https://doi.org/10.1016/S0304-3975(96)00171-5).
- [KOV96] Richard Kennaway, Vincent van Oostrom, and Fer-Jan de Vries. “Meaningless terms in rewriting”. In: *Algebraic and Logic Programming*. 1996, pp. 254–268. DOI: [10.1007/3-540-61735-3\\_17](https://doi.org/10.1007/3-540-61735-3_17).
- [Kur+13] Alexander Kurz et al. “Nominal Coalgebraic Data Types with Applications to Lambda Calculus”. In: *Logical Methods in Computer Science* 9.4 (2013). DOI: [10.2168/lmcs-9\(4:20\)2013](https://doi.org/10.2168/lmcs-9(4:20)2013).
- [Sau10] Alexis Saurin. “Standardization and Böhm Trees for  $\Lambda\mu$ -Calculus”. In: *Functional and Logic Programming (FLOPS 2010)*. 2010, pp. 134–149. DOI: [10.1007/978-3-642-12251-4\\_11](https://doi.org/10.1007/978-3-642-12251-4_11).
- [SV05a] Paula Severi and Fer-Jan de Vries. “Continuity and Discontinuity in Lambda Calculus”. In: *Typed Lambda Calculi and Applications (TLCA 2005)*. 2005, pp. 369–385. DOI: [10.1007/11417170\\_27](https://doi.org/10.1007/11417170_27).
- [SV05b] Paula Severi and Fer-Jan de Vries. “Order Structures on Böhm-Like Models”. In: *Computer Science Logic (CSL 2005)*. 2005, pp. 103–118. DOI: [10.1007/11538363\\_9](https://doi.org/10.1007/11538363_9).
- [SV11a] Paula Severi and Fer-Jan de Vries. “Decomposing the Lattice of Meaningless Sets in the Infinitary Lambda Calculus”. In: *WoLLIC 2011*. 2011, pp. 210–227. DOI: [10.1007/978-3-642-20920-8\\_22](https://doi.org/10.1007/978-3-642-20920-8_22).
- [SV11b] Paula Severi and Fer-Jan de Vries. “Weakening the Axiom of Overlap in Infinitary Lambda Calculus”. In: *22nd International Conference on Rewriting Techniques and Applications (RTA 2011)*. 2011, pp. 313–328. DOI: [10.4230/LIPIcs.RTA.2011.313](https://doi.org/10.4230/LIPIcs.RTA.2011.313).
- [VA19] Lionel Vaux. “Normalizing the Taylor expansion of non-deterministic  $\lambda$ -terms, via parallel reduction of resource vectors”. In: *Logical Methods in Computer Science* 15.3 (2019), 9:1–9:57. DOI: [10.23638/LMCS-15\(3:9\)2019](https://doi.org/10.23638/LMCS-15(3:9)2019).

# Deciding pattern completeness in non-deterministic polynomial time\*

René Thiemann<sup>1</sup> and Akihisa Yamada<sup>2</sup>

<sup>1</sup> University of Innsbruck, Innsbruck, Austria

<sup>2</sup> National Institute of Advanced Industrial Science and Technology, Tokyo, Japan

## Abstract

Pattern completeness is the property that the left-hand sides of a functional program or term rewrite system cover all cases w.r.t. pattern matching. This or related properties are required, if one wants to perform ground confluence proofs by rewriting induction. In order to certify such confluence proofs, we develop a novel algorithm that decides pattern completeness. The algorithm has an asymptotic optimal complexity, as it belongs to the complexity class co-NP. It has been verified in Isabelle/HOL and outperforms existing algorithms, even including the pattern completeness check of the GHC Haskell compiler.

## 1 Introduction

Consider programs written in a declarative style such as functional programs or term rewrite systems (TRSs), where evaluation is triggered by pattern matching. In many applications it is important to ensure that evaluation of a given program cannot get stuck. For instance in Isabelle/HOL [8], in a function definition the patterns must cover all cases, since HOL is a logic of total functions. Moreover, automated theorem proving methods that are based on rewriting induction [1, 9] require similar completeness results, e.g., for proving ground confluence.

**Example 1.** Let  $\mathcal{C}_{\mathbb{N}} = \{\text{true} : \mathbb{B}, \text{false} : \mathbb{B}, 0 : \mathbb{N}, s : \mathbb{N} \rightarrow \mathbb{N}\}$  be the set of constructors to represent the Booleans and natural numbers in Peano's notation. We consider a TRS that defines a function  $\text{even} : \mathbb{N} \rightarrow \mathbb{B}$  to compute whether a natural number is even.

$$\text{even}(0) \rightarrow \text{true} \qquad \text{even}(s(0)) \rightarrow \text{false} \qquad \text{even}(s(s(x))) \rightarrow \text{even}(x) \qquad (1)$$

This TRS is pattern complete, since no matter which number  $n$  we provide as argument, one of the left-hand sides (lhs) will match the term  $\text{even}(n)$ . Note the importance of sorts (types): without them, the evaluation of the (unsorted) term  $\text{even}(s(\text{true}))$  would get stuck.

Kapur et al. [4] proved the decidability of quasi-reducibility, which implies that pattern completeness is also decidable. The algorithm their proof uses has an exponential best-case complexity; i.e., to ensure completeness, one always has to enumerate exponentially many terms and test whether their evaluation does not get stuck. Therefore, Lazrek, Lescanne and Thiel developed the more practical *complement algorithm* [6], which is a decision procedure for pattern completeness in the left-linear case, but may fail otherwise. Pattern completeness of left-linear inputs can also be encoded into a problems on tree automata.

In this paper, we present an algorithm for pattern completeness with the following features: it is a decision procedure, even in the non-linear case; the algorithm is contained in co-NP and

---

\*This research was supported by the Austrian Science Fund (FWF) project I 5943.

is therefore asymptotically optimal; in our experiments it outperforms existing other implementations for checking pattern completeness, including the pattern completeness check of the `ghc` Haskell compiler; and its correctness is fully verified in Isabelle/HOL.

A preliminary version of the algorithm has already been published at FSCD [12]. However, the FSCD algorithm exhibits exponential time and space requirements for non-linear inputs.

The current paper has been submitted in an extended version for a journal publication. The formalization, the executable code and details on the experiments are available at [http://cl-informatik.uibk.ac.at/software/ceta/experiments/pat\\_complete/](http://cl-informatik.uibk.ac.at/software/ceta/experiments/pat_complete/).

## 2 Decision Procedure for Pattern Completeness

We assume familiarity with notions and notations of term rewriting [?]. We consider first-order sorted rewriting. Here,  $\mathcal{S}$  is a set of sorts  $\iota, \iota', \dots$ , and  $\mathcal{V}$  is an infinite set of sorted variables, and  $\mathcal{F}$  is a signature consisting of sorted function symbols  $f : \iota_1 \times \dots \times \iota_n \rightarrow \iota_0 \in \mathcal{F}$ . We write  $t : \iota \in \mathcal{T}(\mathcal{F}, \mathcal{V})$  if  $t$  is a first-order term over  $\mathcal{F}$  and  $\mathcal{V}$  with sort  $\iota$ . We denote  $\mathcal{T}(\mathcal{F}, \mathcal{V})$  for the set of sorted terms over  $\mathcal{F}$  and  $\mathcal{V}$  and  $\mathcal{T}(\mathcal{F})$  for the set of sorted ground terms. Term  $t$  is linear if no variable occurs more than once in  $t$ . A substitution is a sort-preserving map  $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V})$ , and  $t\sigma$  is obtained from term  $t$  by replacing all  $x$  by  $\sigma(x)$ . A term  $\ell$  matches a term  $t$  if  $\ell\sigma = t$  for some substitution  $\sigma$ . The signature  $\mathcal{F}$  is split into two disjoint signatures  $\mathcal{C}$  and  $\mathcal{D}$ , where  $\mathcal{C}$  contains *constructor* symbols and  $\mathcal{D}$  contains *defined* symbols. The *cardinality*  $|\iota|$  of a sort  $\iota \in \mathcal{S}$  is defined as the number of *values* of that sort, i.e.,  $|\{t \mid t : \iota \in \mathcal{T}(\mathcal{C})\}|$ . We assume that every sort is *inhabited*, i.e.,  $|\iota| > 0$ . A sort  $\iota$  is *finite* if  $|\iota|$  is finite, and is infinite otherwise.

Before we design the decision procedure for pattern completeness we first reformulate and generalize this notion, leading to matching problems and pattern problems.

**Definition 2** (Matching Problem and Pattern Problem). *A matching atom is a pair of terms written  $t \sim \ell$ , where  $t$  is called matchee and  $\ell$  is called pattern. The set of matching atoms is denoted by  $\mathcal{M}$ . A finite set  $mp \in \mathcal{P}(\mathcal{M})$  of matching atoms is called a matching problem, and is complete w.r.t. a constructor ground substitution  $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{C})$  if there is some substitution  $\gamma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F})$  such that  $t\sigma = \ell\gamma$  for all  $t \sim \ell \in mp$ . A finite set  $pp \in \mathcal{P}(\mathcal{P}(\mathcal{M}))$  of matching problems is called a pattern problem, and is complete if for each  $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{C})$  there is some  $mp \in pp$  such that  $mp$  is complete w.r.t.  $\sigma$ .*

A set  $L$  of terms *covers* [2, 5, 11] a set  $T$  of terms, if every constructor ground instance of a term in  $T$  is matched by some term in  $L$ . Clearly,  $L$  covers  $T$  if and only if all pattern problems in  $P = \{\{t \sim \ell\} \mid \ell \in L\} \mid t \in T\}$  are complete. The complement algorithm decides whether  $L$  covers  $T$  if all terms in  $L$  are linear. Also pattern completeness is a special case: A program with lhss  $L$  is pattern complete, if every *basic* ground term  $f(t_1, \dots, t_n)$ , where  $f : \iota_1 \times \dots \times \iota_n \rightarrow \iota_0 \in \mathcal{D}$  and  $t_1 : \iota_1, \dots, t_n : \iota_n \in \mathcal{T}(\mathcal{C})$ , is matched by some  $\ell \in L$ . Clearly, this is exactly that  $L$  covers  $\{f(x_1, \dots, x_n) \mid f : \iota_1 \times \dots \times \iota_n \rightarrow \iota_0 \in \mathcal{D}\}$  where  $x_1, \dots, x_n$  are distinct variables.

The proposed decision procedure works in three phases. The upcoming definition shows the inference rules of the first phase. Similar inference rules have already been presented in the FSCD paper [12], but there is a difference in the kind of non-determinism. In the FSCD paper only a “don’t care” kind of non-determinism was present; i.e., there might be several applicable rules, and it does not matter in which order the rules are applied. In contrast, here we have another form of non-determinism, which will be essential for the complexity analysis. The non-deterministic choice in the inference rules of  $\Rightarrow_{nd}$  will need to guess which new problem

is incomplete, if there is any. In the inference rules we also make use of a special matching problem  $\perp_{mp}$  that represents an incomplete matching problem.

**Definition 3** (Phase One of Reduction Rules). *We define the reduction relation  $\rightarrow$  of matching problems by the following rules.*

$$\begin{aligned}
\{f(t_1, \dots, t_n) \sim f(\ell_1, \dots, \ell_n)\} \uplus mp &\rightarrow \{t_1 \sim \ell_1, \dots, t_n \sim \ell_n\} \cup mp && \text{(decompose)} \\
\{t \sim x\} \uplus mp &\rightarrow mp && \text{if } \forall s \sim \ell \in mp. x \notin \text{Var}(\ell) \quad \text{(match)} \\
\{f(\dots) \sim g(\dots)\} \uplus mp &\rightarrow \perp_{mp} && \text{if } f \neq g \quad \text{(clash)} \\
\{t \sim x, t' \sim x\} \uplus mp &\rightarrow \perp_{mp} && \text{if } t \text{ and } t' \text{ do not unify} \quad \text{(clash')} \\
\{y \sim x, t \sim x\} \uplus mp &\rightarrow \perp_{mp} && \text{if } t \notin \mathcal{T}(\mathcal{C}, \mathcal{V}) \quad \text{(no-constructor)} \\
\{f(t_{i1}, \dots, t_{in}) \sim x \mid i \in \{0, \dots, m\}\} \uplus mp &\rightarrow \{t_{ij} \sim z_j \mid i \in \{0, \dots, m\}, j \in \{1, \dots, n\}\} \cup mp \\
&\text{if } x \text{ does not occur in } mp \text{ and } z_1, \dots, z_n \text{ are distinct fresh variables} && \text{(decompose')}
\end{aligned}$$

We further define the relation  $\Rightarrow$  from pattern problems to sets of pattern problems as follows:

$$\begin{aligned}
\{mp\} \uplus pp &\Rightarrow \{\{mp'\} \cup pp\} && \text{if } mp \rightarrow mp' \quad \text{(simp-mp)} \\
\{mp\} \uplus pp &\Rightarrow \{pp\} && \text{if } mp \rightarrow \perp_{mp} \quad \text{(remove-mp)} \\
\{\emptyset\} \uplus pp &\Rightarrow \emptyset && \text{(success)} \\
pp &\Rightarrow \text{Inst}(pp, x) && \text{if } mp \in pp \text{ and } x \sim f(\dots) \in mp \quad \text{(instantiate)} \\
pp \uplus pp' &\Rightarrow \{pp'\} && \text{if } pp \neq \emptyset, \text{ all variables in } pp' \text{ are of finite sort, and} \\
&\forall mp \in pp. \exists x, y, t, \iota. \{y \sim x, t \sim x\} \subseteq mp \wedge y : \iota \in \mathcal{V} \wedge y \neq t \wedge |\iota| \geq \omega \quad \text{(inf-diff)}
\end{aligned}$$

Here, for a pattern problem  $pp$  and a variable  $x : \iota_0 \in \mathcal{V}$ , the pattern problem set  $\text{Inst}(pp, x)$  consists of a pattern problem  $pp\sigma_{x,c} = \{\{t\sigma_{x,c} \sim \ell \mid t \sim \ell \in mp\} \mid mp \in pp\}$  for each  $c : \iota_1 \times \dots \times \iota_n \rightarrow \iota_0 \in \mathcal{C}$ , where  $\sigma_{x,c} = [x \mapsto c(x_1, \dots, x_n)]$  for distinct fresh variables  $x_1 : \iota_1, \dots, x_n : \iota_n \in \mathcal{V}$ .

We finally define the non-deterministic relation  $\Rightarrow_{nd}$  by  $pp \Rightarrow_{nd} pp'$  iff  $pp \Rightarrow P$  and  $pp' \in P$ .

Rules **(decompose)**, **(match)** and **(clash)** correspond to a standard matching algorithm, and **(simp-mp)** and **(remove-mp)** lift these simplifications on matching problems to pattern problems. The **(success)** rule identifies solved matching problems in which case the whole pattern problem is pattern complete, so no new problems are generated. The core rule is **(instantiate)**, where a matching algorithm would detect a failure since non-variable pattern  $f(\dots)$  does not match a variable  $x$ . In contrast,  $x$  in our setting represents an arbitrary constructor ground term. So we just do case analysis, by replacing  $x : \iota_0 \in \mathcal{V}$  by all possible constructor terms of shape  $c(x_1, \dots, x_n)$  for all  $c : \iota_1 \times \dots \times \iota_n \rightarrow \iota_0 \in \mathcal{C}$ .

The remaining four rules are for non-linear problems, where a matching problem contains multiple matching atoms  $t_i \sim x$  for shared pattern variable  $x$ . Such a matching problem restricts to a constructor ground substitution  $\sigma$  such that all  $t_i\sigma$  result in the same term. Hence, **(decompose')** allows decomposing  $x$  if all such  $t_i$  have the same root symbol. Otherwise, a failure is reported in **(clash')**, if there are  $t_i$  and  $t_j$  that are not unifiable. Moreover, if  $t_i$  is a variable  $y$ , then  $t_i\sigma$  is a constructor ground term, and thus, a failure is reported in **(no-constructor)** whenever  $t_j \notin \mathcal{T}(\mathcal{C}, \mathcal{V})$ . Finally, there is a potential difference  $y \neq t$  in **(inf-diff)**. Here, one can prove that such a matching problem can never cover all possible constructor ground substitutions  $\sigma$ , if  $y$  has an infinite type.

The following theorem states that  $\Rightarrow_{nd}$  can be used as a first step to decide completeness of pattern problems, which simplifies arbitrary pattern problems to *finite constructor form*. A pattern problem  $pp$  has this form, if for all  $t \sim \ell \in mp \in pp$ ,  $\ell$  and  $t$  have a finite sort,  $\ell$  is a variable and  $t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ .

**Theorem 4** (Phase One: Switching to Finite Constructor Form).

- The maximum number of  $\Rightarrow_{nd}$ -steps is polynomially bounded.
- If  $pp \Rightarrow_{nd}^! pp'$  then  $pp'$  is in finite constructor form.
- $pp$  is incomplete iff there is some  $pp'$  such that  $pp \Rightarrow_{nd}^! pp'$  and  $pp'$  is incomplete.

Note that in case of linear pattern problems, the first phase is already a decision procedure: then  $pp$  is incomplete iff  $pp \Rightarrow_{nd}^! \emptyset$ . In this case the only normal forms are  $\emptyset$ —representing an incomplete pattern problem—and  $\{\emptyset\}$ —representing a trivially complete pattern problem.

**Example 5.** Consider a signature with sorts  $\{\iota_0, \dots, \iota_n\}$  where each sort has a single constructor:  $c_0 : \iota_0$  and  $c_{i+1} : \iota_i \times \iota_i \rightarrow \iota_{i+1}$  for all  $i < n$ . Then every  $\iota_i$  contains exactly one ground term  $gt_i$ , which is a full binary tree of depth  $i$ , i.e., the size of  $gt_i$  is  $2^i - 1$ .

Next consider the TRS over this signature that has left-hand sides  $f(x_0, c_1(x_0, x_0), -, -, \dots)$ ,  $f(-, x_1, c_2(x_1, x_1), -, -, \dots)$ ,  $\dots$ ,  $f(-, \dots, -, x_{n-1}, c_n(x_{n-1}, x_{n-1}))$  where each underscore represents a fresh variable, and the aim is to check whether the TRS is pattern complete. The initial pattern problem encoding this problem is thus

$$pp_0 = \{\{t \sim f(x_0, c_1(x_0, x_0), -, -, \dots)\}, \{t \sim f(-, x_1, c_1(x_1, x_1), -, -, \dots)\}, \dots\}.$$

where  $t = f(y_0, y_1, y_2, \dots, y_n)$ . The first phase performs the following steps on  $pp_0$ .

$$\begin{aligned} pp_0 &\Rightarrow_{nd}^* \{\{y_0 \sim x_0, y_1 \sim c_1(x_0, x_0)\}, \{y_1 \sim x_1, y_2 \sim c_2(x_1, x_1)\}, \{y_2 \sim x_2, y_3 \sim c_3(x_2, x_2)\}, \dots\} \\ &\Rightarrow_{nd}^! \{\{y_0 \sim x_0, z_1 \sim x_0, z_2 \sim x_0\}, \{c_1(z_1, z_2) \sim x_1, z_3 \sim x_1, z_4 \sim x_1\}, \\ &\quad \{c_2(z_3, z_4) \sim x_2, z_5 \sim x_2, z_6 \sim x_2\}, \dots\} =: pp_1 \end{aligned}$$

The FSCD algorithm would now fully instantiate all variables  $y_0, z_1, z_2, \dots$  in  $pp_1$  to the corresponding ground terms  $gt_i$  in order to finally detect pattern completeness. This process requires exponential time and space.

In the non-linear case we still have to solve problems in finite constructor form, which will be done in phases two and three of the new algorithm. Since the first phase yields normal forms whose patterns are only variables, phase two starts with simplifying the presentation so that names of the pattern variables are irrelevant. For instance, we identify the matching problem  $\{t_1 \sim x, t_2 \sim y, t_3 \sim x, t_4 \sim x, t_5 \sim y\}$  with  $\{\{t_1, t_3, t_4\}, \{t_2, t_5\}\}$ , forgetting the variable symbols  $x$  and  $y$  by grouping all corresponding matchee terms in equivalence classes. From now on we just consider finite constructor problems in this simplified form. For instance,  $pp_1$  in Example 5 becomes:

$$\{\{\{y_0, z_1, z_2\}\}, \{\{c_1(z_1, z_2), z_3, z_4\}\}, \{\{c_2(z_3, z_4), z_5, z_6\}\}, \dots\}$$

The second phase eliminates all occurrences of terms that are not variables.

**Definition 6** (Transformation to Finite Variable Form). A matching problem  $mp$  is in finite variable form if each  $e \in mp$  contains only variables of the same finite sort. A pattern problem  $pp$  is in finite variable form if all matching problems  $mp \in pp$  are. We define  $\xrightarrow{s}$  as binary relation on simplified matching problems, consisting of simplified forms of (*decompose*) and (*clash*'), and additionally the following:

$$\begin{aligned} \{e\} \uplus mp &\xrightarrow{s} mp && \text{if } |e| = 1 && (\text{unique}) \\ \{e\} \uplus mp &\xrightarrow{s} mp && \text{if the terms in } e \text{ have sort } \iota \text{ and } |\iota| = 1 && (\text{card-1}) \end{aligned}$$



We define  $\overset{s}{\Rightarrow}$ , similarly to  $\Rightarrow$ , consisting of simplified forms of (*simpl-mp*), (*remove-mp*), and (*success*), and additionally the following rules:

$$pp \overset{s}{\Rightarrow} \text{Inst}(pp, x) \quad \text{if } \{\{x, t\}\} \in pp \text{ and } t \text{ is not a variable} \quad (\text{instantiate'})$$

$$pp \uplus \{\{\{t, t'\} \uplus e\} \uplus mp\} \overset{s}{\Rightarrow} \{\{\{t, t'\}\} \cup pp, \{\{\{t'\} \cup e\} \cup mp\} \cup pp\} \quad \text{if } e \neq \emptyset \text{ or } mp \neq \emptyset, \text{ and exactly one of } t \text{ and } t' \text{ is a variable} \quad (\text{split})$$

$$pp \uplus \{\{\{x_1, t_1\} \uplus e_1\} \uplus mp_1, \dots, \{\{x_n, t_n\} \uplus e_n\} \uplus mp_n\} \overset{s}{\Rightarrow} \{pp\} \quad \text{if } n > 0, x_i \neq t_i \text{ for each } i, x_1, \dots, x_n, t_1, \dots, t_n \text{ are of sort } \iota \text{ and not in } \text{Var}(pp), |\{t_1, \dots, t_n\}| < |\iota| \quad (\text{card-large})$$

Finally, we define the non-deterministic relation  $\overset{s}{\Rightarrow}_{nd}$ :  $pp \overset{s}{\Rightarrow}_{nd} pp'$  iff  $pp \overset{s}{\Rightarrow} P$  and  $pp' \in P$ .

**Lemma 7** (Phase Two: Switching to Finite Variable Form).

- The number of  $\overset{s}{\Rightarrow}_{nd}$ -steps is polynomially bounded.
- If  $pp \overset{s}{\Rightarrow}_{nd}^! pp'$  then  $pp'$  is in finite variable form.
- $pp$  is incomplete iff there is some  $pp'$  such that  $pp \overset{s}{\Rightarrow}_{nd}^! pp'$  and  $pp'$  is incomplete.

Note that without the splitting and the restriction on the (*instantiate'*) rule we would not achieve a polynomial bound: For a relaxed version where instantiation is allowed as soon as some equivalence class contains two terms  $x$  and  $t$  where  $t$  is not a variable, one obtains an exponential time and exponential space algorithm. In particular,  $pp_1$  in Example 5 can start an exponentially long derivation with  $\overset{s}{\Rightarrow}_{nd}$  using the relaxed version.

In phase three it remains to deal with pattern problems  $pp$  in finite variable form, i.e., every  $e \in mp \in pp$  consists of variables of the same finite sort. Such a problem is pattern complete iff for every substitution  $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{C})$  there exists  $mp \in pp$  such that  $\{x_1, \dots, x_n\} \in mp$  implies  $x_1\sigma = \dots = x_n\sigma$ . At this point the term structure is of no relevance any further, i.e., we can switch from term substitution  $\sigma : \mathcal{V} \rightarrow \mathcal{T}(\mathcal{C})$  to integer assignment  $\alpha : \mathcal{V} \rightarrow \mathbb{Z}$ . One only must constrain the integer assignments to take the cardinalities of the sorts into account.

**Definition 8** (Transformation to SMT Formula). Let  $pp$  be a pattern problem in finite variable form such that  $\text{Var}(pp) = \{y_1 : \iota_1, \dots, y_n : \iota_n\}$ . We define the encoding  $\psi(pp)$  of  $pp$  as

$$\psi(pp) := \left( \bigvee_{mp \in pp} \bigwedge_{\{x_1, \dots, x_n\} \in mp} x_1 = \dots = x_n \right) \wedge \bigwedge_{k=1}^n 1 \leq y_k \leq |\iota_k|.$$

To finalize the decision procedure, we just need some algorithm to decide the validity of formula  $\psi(pp)$ . Note that  $\psi(pp)$  can be expressed in well known logics such as integer difference logic, linear integer arithmetic, or bit-vector arithmetic. It is well known that deciding validity in each of these three logics is co-NP complete.

**Lemma 9** (Phase Three: Deciding Completeness via SMT Solving). A pattern problem  $pp$  in finite variable form is pattern complete iff  $\psi(pp)$  is valid in  $\mathbb{Z}$ , and the validity is in co-NP.

**Theorem 10.** Pattern completeness is decidable in non-deterministic polynomial time.

Note that to arrive at Theorem 10 it does not suffice to just link the results of each phase together. One also has to be careful about the arithmetic operations involved. In particular, a direct computation of  $|\iota|$  is too expensive, as  $|\iota|$  might be double exponential in the number



of constructors. For instance, if in Example 5 one adds a second constructor  $d : \iota_0$  to  $\iota_0$ , then  $|\iota_0| = 2$ ,  $|\iota_1| = |\iota_0|^2 = 4$ ,  $|\iota_2| = |\iota_1|^2 = 16$ ,  $\dots$ ,  $|\iota_n| = 2^{2^n}$ .

Here, the key is to compute  $\min(2^{64}, |\iota|)$  instead of  $|\iota|$  and then adjust the inference rules and the implementation to work with these approximations of  $|\iota|$ . For instance, condition  $|\{t_1, \dots, t_n\}| < |\iota|$  in (card-large) can be replaced by the sufficient criterion  $|\{t_1, \dots, t_n\}| < \min(2^{64}, |\iota|)$ . Note that in all practical cases, this sufficient criterion is precise enough, since otherwise one would have to treat simplified pattern problems with at least  $2^{64}$  many terms.

### 3 Experiments

A full evaluation of the performance on linear pattern problems is already contained in the FSCD paper. Since the new algorithm and the FSCD algorithm do not differ in their treatment of linear problems, we just repeat the FSCD results here: our algorithm is significantly faster than the following alternatives: the complement algorithm (implemented in AGCP [1]), a tree automaton encoding (using FORT-h [7] as backend), and the pattern completeness check of the ghc Haskell compiler.

For the non-linear case we utilized 300 pattern complete inputs (YES) and 300 incomplete inputs (NO). The performance of the FSCD algorithm (FSCD) and the new algorithm of this paper (NEW) is summarized in the table on the right. It displays the number of successful runs with a timeout of 60 seconds for each input problem.

	YES	NO
FSCD algorithm	120	123
NEW algorithm	211	300

The overall interpretation of the data is easy: NEW clearly outperforms FSCD. As a matter of fact, the example inputs contain many pattern problems that have been detected as counter-examples to membership in co-NP for intermediate versions of our algorithm. These examples—including Example 5—were produced when trying to optimize the FSCD algorithm into one that has strong theoretical bounds, such as NEW. The 89 problems that NEW could not solve within the time limit are all encodings of the pigeon hole principle.

For further details on the experiments we refer to the website with supplementary material.

### 4 Future Work

We see some opportunities for future work. First, one can integrate an improved strategy to select variables for instantiation, in particular since permutations in the input cause severe differences in runtime. One can also try to further improve the implementation, e.g., by following suggestions of Sestoft [10, Section 7.5] such as the integration of memoization. Second, one might add counter-example generation into the formalization and into the verified implementation. Third, it remains open whether a similar syntax directed decision procedure for quasi-reducibility can be designed, i.e., where matching may occur in arbitrary subterms. Fourth, one might consider an extension where it is allowed to add structural axioms to some symbols such as associativity and commutativity. And finally, one can utilize this decision algorithm, e.g., in order to develop a verified checker for ground confluence proofs for pattern complete TRSs [1].

**Acknowledgments** We thank the anonymous reviewers for their constructive feedback.

## References

- [1] Takahito Aoto and Yoshihito Toyama. Ground confluence prover based on rewriting induction. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPICs*, pages 33:1–33:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- [2] Hubert Comon. Sufficient completeness, term rewriting systems and "anti-unification". In Jörg H. Siekmann, editor, *8th International Conference on Automated Deduction, Oxford, England, July 27 - August 1, 1986, Proceedings*, volume 230 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 1986.
- [3] Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. SAT competition 2020. *Artif. Intell.*, 301:103572, 2021.
- [4] Deepak Kapur, Paliath Narendran, and Hantao Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987.
- [5] Alain Laville. Lazy pattern matching in the ML language. In Kesav V. Nori, editor, *Foundations of Software Technology and Theoretical Computer Science, Seventh Conference, Pune, India, December 17-19, 1987, Proceedings*, volume 287 of *Lecture Notes in Computer Science*, pages 400–419. Springer, 1987.
- [6] Azeddine Lazrek, Pierre Lescanne, and Jean-Jacques Thiel. Tools for proving inductive equalities, relative completeness, and omega-completeness. *Inf. Comput.*, 84(1):47–70, 1990.
- [7] Aart Middeldorp, Alexander Lochmann, and Fabian Mitterwallner. First-order theory of rewriting for linear variable-separated rewrite systems: Automation, formalization, certification. *J. Autom. Reason.*, 67(2):14, 2023.
- [8] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.
- [9] Uday S. Reddy. Term rewriting induction. In Mark E. Stickel, editor, *10th International Conference on Automated Deduction, Kaiserslautern, FRG, July 24-27, 1990, Proceedings*, volume 449 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 1990.
- [10] Peter Sestoft. MK pattern match compilation and partial evaluation. In Olivier Danvy, Robert Glück, and Peter Thiemann, editors, *Partial Evaluation, International Seminar, Dagstuhl Castle, Germany, February 12-16, 1996, Selected Papers*, volume 1110 of *Lecture Notes in Computer Science*, pages 446–464. Springer, 1996.
- [11] Jean-Jacques Thiel. Stop losing sleep over incomplete data type specifications. In Ken Kennedy, Mary S. Van Deusen, and Larry Landweber, editors, *Conference Record of the Eleventh Annual ACM Symposium on Principles of Programming Languages, Salt Lake City, Utah, USA, January 1984*, pages 76–82. ACM Press, 1984.
- [12] René Thiemann and Akihisa Yamada. A verified algorithm for deciding pattern completeness. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction, FSCD 2024, July 10-13, 2024, Tallinn, Estonia*, volume 299 of *LIPICs*, pages 27:1–27:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

# Proving and disproving feasibility with `infChecker`\*

Raúl Gutiérrez and Salvador Lucas

DSIC & VRain, Universitat Politècnica de València, Spain  
{rgutierrez,slucas}@dsic.upv.es

## Abstract

Given a first-order theory  $\text{Th}$ , we say that a boolean combination  $F$  of atoms is *Th-feasible* if there is a substitution  $\sigma$  such that  $\sigma(F)$  is deducible from  $\text{Th}$ , i.e.,  $\text{Th} \vdash \sigma(F)$  holds. Otherwise,  $F$  is *infeasible*. In the realm of (conditional) term rewriting, many interesting problems can be treated as (in)feasibility problems: joinability of terms (and critical pairs), reachability, reducibility, etc. This paper shows how general feasibility problems can be handled now with the new version of the tool `infChecker`.

## 1 Introduction

Given terms  $s$  and  $t$ , the notion of “feasibility of a reachability problem”  $s \rightarrow^* t$  is well-known in term rewriting. In this setting, instead of just checking whether  $s \rightarrow_{\mathcal{R}}^* t$  (reachability test), it is often useful to check whether  $\sigma(s) \rightarrow_{\mathcal{R}}^* \sigma(t)$  holds *for some substitution*  $\sigma$ . This is often called a *feasibility* test (for reachability). Accordingly, if no substitution  $\sigma$  satisfies the requirement, then  $s \rightarrow^* t$  is said to be *infeasible*. This problem often arises when considering conditions  $s \approx t$  in the conditional part  $c$  of the rules  $\ell \rightarrow r \Leftarrow c$  of an *oriented* Conditional Term Rewriting System  $\mathcal{R}$ , where  $\approx$  is interpreted as many-step rewriting, i.e.,  $\rightarrow^*$ . If the sequence  $s_1 \approx t_1, \dots, s_n \approx t_n$  of conditions of  $c$  is proved *infeasible* (i.e., no substitution  $\sigma$  simultaneously satisfies all of them), then the rule itself is said to be *infeasible* and it does *not* contribute to the rewriting relation  $\rightarrow_{\mathcal{R}}$  of  $\mathcal{R}$ . For some purposes, e.g., for proving *confluence* or *termination* of  $\mathcal{R}$ , the rule can safely be *removed* from  $\mathcal{R}$ . Thus, a number of tools have been developed to prove *infeasibility* of a sequence of reachability problems, see [9, 10, 12, 14, 15] and also [11].

However, other interpretations of ‘ $\approx$ ’ are used in, e.g., *join* or *semi-equational* CTRSs, where conditions  $s \approx t$  are treated as joinability  $s \downarrow t$  or conversion  $s \leftrightarrow^* t$ , respectively [13, Definition 7.1.13]. In these cases, conditional rules can also be *infeasible*, but reachability tests are not always useful to prove it. In order to overcome this problem, in [3] we rely on first-order logic to provide a general notion of

- *f-condition* (an atomic formula, e.g.,  $s \rightarrow^* t$ ,  $s \downarrow t$ ,  $s \leftrightarrow^* t, \dots$ ),
- *f-sequence* (of f-conditions, interpreted as a *conjunction of atoms*), and
- *f-goal* (interpreted as *disjunctions of f-sequences*).

These *expressions*  $F$  (viewed as boolean combinations of atoms as explained above) are said to be *feasible* with respect to a given first-order theory  $\text{Th}$  if there is a substitution  $\sigma$  such that  $\text{Th} \vdash \sigma(F)$  holds. Otherwise,  $F$  is said to be *infeasible* [3, Definition 1]. In this setting, the (in)feasibility of sequences of goals  $s \rightarrow^* t$ ,  $s \downarrow t$ , or  $s \leftrightarrow^* t$  can be homogeneously investigated if predicate symbols  $\rightarrow^*$ ,  $\downarrow$ ,  $\leftrightarrow^*$ , etc., are defined by first-order sentences included in  $\text{Th}$ .

---

\*Partially supported by MCIN/AEI project PID2021-122830OB-C42 funded by MCIN/AEI/10.13039/501100011033 and by “ERDF A way of making Europe” and by the grant CIPROM/2022/6 funded by Generalitat Valenciana.

Table 1: Generic sentences of the first-order theory of a GTRS

Label	Sentence
$(\text{Rf})^{\bowtie}$	$(\forall x) x \bowtie x$
$(\text{Co})^{\bowtie, \bowtie'}$	$(\forall x, y, z) x \bowtie y \wedge y \bowtie' z \Rightarrow x \bowtie' z$
$(\text{Pr})_{f,i}^{\bowtie}$	$(\forall x_1, \dots, x_k, x'_i) x_i \bowtie x'_i \Rightarrow f(x_1, \dots, x_i, \dots, x_k) \bowtie f(x_1, \dots, x'_i, \dots, x_k)$
$(\text{HC})_{A \Leftarrow A_1, \dots, A_n}$	$(\forall x_1, \dots, x_p) A_1 \wedge \dots \wedge A_n \Rightarrow A$ where $x_1, \dots, x_p$ are the variables occurring in $A_1, \dots, A_n$ and $A$

In this paper, we report on the extension of the functionality of `infChecker` to fulfill the theoretical framework in [3] which is sketched above. As described in [3, Section 5], the current implementation of `infChecker` is able to deal with (join, oriented, or semi-equational) CTRSs  $\mathcal{R}$ , possibly including replacement restrictions as given by a *replacement* map  $\mu$ , indicating for each function symbol  $f$  of the signature, the arguments  $\mu(f)$  on which rewritings can be performed [6]. The first-order theory  $\overline{\mathcal{R}}$  associated to  $\mathcal{R}$  is automatically generated by the tool, according to [7, Section 3]. And, besides  $s \rightarrow^* t$ ,  $s \downarrow t$ , or  $s \leftrightarrow^* t$ , `infChecker` is also able to deal with  $s \rightarrow t$  (one-step rewriting),  $s \leftrightarrow t$  (one-step rewriting or inverse rewriting),  $s \triangleright t$  ( $t$  is a strict subterm of  $s$ ), and the corresponding *context-sensitive* versions.

In order to exemplify the new use of the tool in a rewriting-based setting, we use *Generalized Term Rewriting Systems* [7], extending CTRSs by the use of a replacement map  $\mu$  and also by allowing for more general conditions in rules, namely, *atoms defined by a set of Horn clauses*.

## 2 Generalized Term Rewriting Systems

A GTRS is a tuple  $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ , where  $\mathcal{F}$  (resp.  $\Pi$ ) is a signature of *function* (resp. *predicate*) symbols, with  $\rightarrow, \rightarrow^* \in \Pi$ ;  $\mu$  is a *replacement map*;  $H$  is a set of *definite Horn clauses*  $A \Leftarrow c$ , where the predicate symbol of  $A$  is not  $\rightarrow$  or  $\rightarrow^*$ ; and  $R$  is a set of *rewrite rules*  $\ell \rightarrow r \Leftarrow c$  where  $\ell$  is not a variable. In both cases,  $c$  is a sequence of atoms.

**Example 1.** The following GTRS  $\mathcal{R}$  implements a test of evenness/oddity of numbers in Peano's notation through a “labeling” with symbols `even` and `odd` [8, Example 7].

$$\begin{array}{llll}
\text{isEven}(0) & (1) & x + y \rightarrow y + x & (5) \\
\text{isEven}(s(s(n))) \Leftarrow \text{isEven}(n) & (2) & \text{test}(x) \rightarrow \text{even}(x) \Leftarrow \text{isEven}(x) & (6) \\
0 + x \rightarrow x & (3) & \text{test}(x) \rightarrow \text{odd}(x) \Leftarrow \text{isEven}(s(x)) & (7) \\
x + 0 \rightarrow x & (4) & & 
\end{array}$$

Computations with GTRSs  $\mathcal{R}$  are defined by deduction in a FO-theory [7, Section 7.3]:

$$\overline{\mathcal{R}} = \{(\text{Rf})^{\rightarrow^*}, (\text{Co})^{\rightarrow, \rightarrow^*}\} \cup \{(\text{Pr})_{f,i}^{\rightarrow} \mid f \in \mathcal{F}, i \in \mu(f)\} \cup \{(\text{HC})_{\alpha} \mid \alpha \in H \cup R\},$$

where (see Table 1),  $(\text{Rf})^{\rightarrow^*}$  expresses *reflexivity* of many-step rewriting;  $(\text{Co})^{\rightarrow, \rightarrow^*}$  expresses *compatibility* of one-step and many-step rewriting; for each  $k$ -ary function symbol  $f$  and  $i \in \mu(f)$ ,  $(\text{Pr})_{f,i}^{\rightarrow}$  enables the *propagation* of rewriting steps in the  $i$ -th immediate *active* subterm  $t|_i$  of a term  $t$  with root symbol  $f$ ; finally, for each Horn clause  $\alpha \in H \cup R$ ,  $(\text{HC})_{\alpha}$  provides the usual *implicative* form. We often use (numeric or symbolic) labels  $\alpha$  to refer to clauses and rules  $A \Leftarrow c$ , written  $\alpha : A \Leftarrow c$ . We write  $s \rightarrow_{\mathcal{R}} t$  (resp.  $s \rightarrow_{\mathcal{R}}^* t$ ) if  $\overline{\mathcal{R}} \vdash s \rightarrow t$  (resp.

$\overline{\mathcal{R}} \vdash s \rightarrow^* t$ ). Confluence of GTRSs  $\mathcal{R}$  is investigated by means of conditional pairs (with label  $\pi$ )  $\pi : \langle s, t \rangle \Leftarrow c$ , where  $s$  and  $t$  are terms and  $c$  is a sequence of atoms [7, Section 5]. Then,  $\pi$  is

- $\overline{\mathcal{R}}$ -(in)feasible (or just (in)feasible if no confusion arises) if  $c$  is  $\overline{\mathcal{R}}$ -(in)feasible.
- $\overline{\mathcal{R}}$ -joinable (or just joinable) if for all substitutions  $\sigma$ , whenever  $\overline{\mathcal{R}} \vdash \sigma(\gamma)$  holds for all  $\gamma \in c$ , there is  $u$  such that  $\sigma(s) \rightarrow_{\mathcal{R}}^* u$  and  $\sigma(t) \rightarrow_{\mathcal{R}}^* u$ .
- Trivial, if  $s = t$ .

According to [7, Definition 59], given variable disjoint rules  $\alpha : \ell \rightarrow r \Leftarrow c$  and  $\alpha' : \ell' \rightarrow r' \Leftarrow c'$ ,

- $\pi_{\alpha,p,\alpha'}$  denotes the *Conditional Critical Pair* (CCP)  $\langle \theta(\ell[r']_p), \theta(r) \rangle \Leftarrow \theta(c), \theta(c')$ , where  $p \in \text{Pos}_{\mathcal{F}}^{\mu}(\ell)$  is a nonvariable active position of  $\ell$  such that  $\ell|_p$  and  $\ell'$  unify with *mgu*  $\theta$ .
- $\pi_{\alpha,x,p}$  denotes the *Conditional Variable Pair* (CVP)  $\langle \ell[x']_p, r \rangle \Leftarrow x \rightarrow x', c$ , where  $x \in \text{Var}^{\mu}(\ell)$  is an active variable of  $\ell$ ,  $p \in \text{Pos}_x^{\mu}(\ell)$  is an active position of  $x$  in  $\ell$ , and  $x'$  is a fresh variable.

Two terms  $s$  and  $t$  are *strongly joinable* if there are terms  $u$  and  $u'$  such that  $s \rightarrow^= u \xrightarrow{*} u'$  and  $s \rightarrow^* u' \rightarrow^= t$ , where  $\rightarrow^=$  is  $\rightarrow \cup =$  [2, Definition 6.3.2]. Strong joinability of  $s$  and  $t$  is equivalent to the feasibility of the sequence

$$s^{\downarrow} \rightarrow^= z, s^{\downarrow} \rightarrow^* z', t^{\downarrow} \rightarrow^* z, t^{\downarrow} \rightarrow^= z' \quad (8)$$

where  $s^{\downarrow}$  is the *grounded* version of  $s$  obtained by replacing each occurrence of a variable  $x$  in  $s$  by a fresh constant  $c_x$ , see, e.g., [1, page 224], and  $z$  and  $z'$  are fresh variables. As explained in [8, Example 13], confluence of  $\mathcal{R}$  in Example 1 can be proved using [8, Theorem 1] if the following CCPs are *strongly joinable*

$$\pi_{(3),\Lambda,(4)} : \quad \langle 0, 0 \rangle \quad (9)$$

$$\pi_{(3),\Lambda,(5)} : \quad \langle x + 0, x \rangle \quad (10)$$

$$\pi_{(4),\Lambda,(5)} : \quad \langle 0 + x, x \rangle \quad (11)$$

$$\pi_{(6),\Lambda,(7)} : \quad \langle \text{odd}(x), \text{even}(x) \rangle \Leftarrow \text{isEven}(x), \text{isEven}(s(x)) \quad (12)$$

The symmetric versions  $\pi_{(4),\Lambda,(3)}$ ,  $\pi_{(5),\Lambda,(3)}$ ,  $\pi_{(5),\Lambda,(4)}$ , and  $\pi_{(7),\Lambda,(6)}$  of these CCPs are also *strongly joinable* due to the symmetric definition of strong joinability above; thus, we do not explicitly treat them.

**Example 2.** Strong joinability of the (nontrivial) unconditional pairs (10) and (11) is proved by showing feasibility of each of the corresponding sequence (8), i.e.,

$$c_x + 0 \rightarrow^= z, c_x + 0 \rightarrow^* z', c_x \rightarrow^* z, c_x \rightarrow^= z' \quad (13)$$

$$0 + c_x \rightarrow^= z, 0 + c_x \rightarrow^* z', c_x \rightarrow^* z, c_x \rightarrow^= z' \quad (14)$$

Strong joinability of the conditional pair (12) is proved instead by showing the  $\overline{\mathcal{R}}$ -infeasibility of the conditional part, i.e., of

$$\text{isEven}(x), \text{isEven}(s(x)) \quad (15)$$

Two terms  $s$  and  $t$  are *strongly  $\Downarrow$ -joinable* if there are terms  $u, u'$  such that  $s \Downarrow^* u \Downarrow^* u'$  and  $s \Downarrow^* u' \Downarrow^* t$ . We also need to prove that the following CVPs are strongly  $\Downarrow$ -joinable.

$$\pi_{(6),x,1} : \quad \langle \text{test}(x'), \text{even}(x) \rangle \Leftarrow x \rightarrow x', \text{isEven}(x) \quad (16)$$

$$\pi_{(7),x,1} : \quad \langle \text{test}(x'), \text{odd}(x) \rangle \Leftarrow x \rightarrow x', \text{isEven}(s(x)) \quad (17)$$

**Example 3.** Both (16) and (17) are proved strongly  $\Downarrow$ -joinable by showing the infeasibility of the corresponding conditional parts:

$$x \rightarrow x', \text{isEven}(x) \quad (18)$$

$$x \rightarrow x', \text{isEven}(s(x)) \quad (19)$$

In the following sections, we show how the new features added to infChecker are used to obtain a mechanized proof of confluence of  $\mathcal{R}$  in Example 1.

### 3 Input Format

To describe our theories in infChecker, we use the old block-based COPS format<sup>1</sup>, extending it to handle the new capabilities of our tool.

A problem described using the original block-based COPS format includes a **PROBLEM** block, which allows us to identify that we are dealing with an infeasibility problem rather than a confluence problem; a **CONDITIONTYPE** block, if conditional rules are present, which lets us choose whether the rewrite relation in the conditions is oriented, semi-equational, or join; a **VAR** block, which identifies the variables of our rewrite system; a **RULES** block, which describes the (possibly conditional) rules of our system; and finally, the infeasibility conditions to be analyzed, consisting of two blocks: a **VAR** block that specifies the variables used in the conditions and a **CONDITION** block with the infeasibility condition.

Additionally, in the previous version of our tool [3], we included the possibility of describing the conditions of the rules using the following relations on conditions: one rewriting step ( $\rightarrow$ ), one CS-rewriting step ( $\backslash \rightarrow$ ), zero or more rewriting steps ( $\rightarrow^*$ ), zero or more CS-rewriting steps ( $\backslash \rightarrow^*$ ), one or more rewriting steps ( $\rightarrow^+$ ), one or more CS-rewriting steps ( $\backslash \rightarrow^+$ ), subterm ( $| \geq$ ) and strict subterm ( $| >$ ), joinability ( $\rightarrow^* \leftarrow$ ), CS-joinability ( $\backslash \rightarrow^* \leftarrow$ ), one convertibility step ( $\leftrightarrow$ ), one CS-convertibility step ( $\leftarrow \backslash \rightarrow$ ), zero or more convertibility steps ( $\leftrightarrow^*$ ) and zero or more CS-convertibility steps ( $\leftarrow \backslash \rightarrow^*$ ). Furthermore, for teaching purposes, we recently added the following relations: zero or one rewriting step ( $\rightarrow =$ ) and zero or one CS-rewriting step ( $\backslash \rightarrow =$ ).

To extend the capabilities of our tool, we have continued expanding the types of conditions we can handle and have introduced three new types of blocks:

- The **EQUATIONS** block allows us to describe equations, which can be conditional. Such conditions can also be equations or any other type of condition accepted by the rules.
- The **HORN-CLAUSES** block allows us to describe predicates as Horn clauses. As with equations, the conditions of our Horn clause can be atoms or any other type of condition accepted by the rules.
- The **FO-THEORY** block allows us to describe a formula in first-order logic using the operators  $\sim$  (not),  $\wedge$  (and),  $\vee$  (or),  $\Rightarrow$  (implies) and  $\Leftrightarrow$  (if and only if). The atoms of the formula can be any type of condition accepted by the rules.
- The system's conditions, besides including the previously described relations, may contain equations and predicates, but not first-order formulas.

The full syntax of our input is described in Figure 1.

```

infproblem ::= (PROBLEM INFEASIBILITY)
              rew-crew [(VAR idlist)] (CONDITION condlist)
              [(COMMENT string)]

rew-crew ::= rew | crew | cs-rew | cs-crew
rew ::= [(VAR idlist)] [(FO-THEORY formulalist)]
        [(HORN-CLAUSES predlist)] [(EQUATIONS eqlist)]
        (RULES rulelist) [(COMMENT string)]
crew ::= (CONDITIONTYPE ctype) [(VAR idlist)]
        [(FO-THEORIES formulalist)] [(HORN-CLAUSES predlist)]
        [(EQUATIONS eqlist)] (RULES crulelist) [(COMMENT string)]
cs-rew ::= [(VAR idlist)] repmap [(FO-THEORIES formulalist)]
          [(HORN-CLAUSES predlist)] [(EQUATIONS eqlist)]
          (RULES rulelist) [(COMMENT string)]
cs-crew ::= (CONDITIONTYPE ctype) [(VAR idlist)] repmap
           [(FO-THEORIES formulalist)] [(HORN-CLAUSES predlist)]
           [(EQUATIONS eqlist)] (RULES crulelist) [(COMMENT string)]

idlist ::= ε | id idlist
repmap ::= (REPLACEMENT-MAP cslist) | (CONTEXTSENSITIVE csstratlist)
rulelist ::= ε | rule rulelist
rule ::= term -> term
crulelist ::= ε | crule rulelist
crule ::= term -> term | term == term | condlist
eqlist ::= ε | eq eqlist
eq ::= term = term | term = term | condlist
ctype ::= SEMI-EQUATIONAL | JOIN | ORIENTED
condlist ::= cond | cond, condlist
cond ::= term == term | term extrel term | term
term ::= id | id(termlist)
termlist ::= term | term, termlist
formulalist ::= formula | formula formulalist
formula ::= pred | ~formula | formula /\ formula | formula \/ formula
           | formula => formula | formula <=> formula
predlist ::= pred | pred predlist
pred ::= term | term | cond
cslist ::= fun | fun cslist
csstratlist ::= ε | (id intlist) csstratlist
intlist ::= ε | int intlist
fun ::= (id intlist)
intlist ::= ε | int, intlist
extrel ::= -> | \-> | ->= | \->= | ->* | \->* | ->+ | \->+ | |>= | |>
           | ->*<- | \->*<- / | <-> | <-/\-> | <->*<- | <-/\->*<- | =

```

Figure 1: Extended COPS format

**Example 4.** The GTRS  $\mathcal{R}$  in Example 1 is encoded as in Figure 2. The feasibility conditions in Examples 2 and 3 as follows: Strong joinability of CCPs (10) and (11) as the feasibility of:

```

(VAR x x' z z')
(CONDITION  add(cX,0) ->= z, add(cX,0) ->* z', cX ->* z, cX ->= z' )

```

and

```

(VAR x x' z z')
(CONDITION  add(0,cX) ->= z, add(0,cX) ->* z', cX ->* z, cX ->= z' )

```

The infeasibility of CCP (12) as the infeasibility of:

---

<sup>1</sup><http://project-coco.uibk.ac.at/problems/>

```

(VAR x y)
(HORN-CLAUSES
  isEven(0)
  isEven(s(s(x))) | isEven(x)
)
(RULES
  add(0,x) -> x
  add(x,0) -> x
  add(x,y) -> add(y,x)
  test(x) -> even(x) | isEven(x)
  test(x) -> odd(x) | isEven(s(x))
)

```

Figure 2: Encoding of the GTRS in Example 1

```

(VAR x)
(CONDITION  isEven(x), isEven(s(x))  )

The infeasibility of CVPs (16) and (17) as as the infeasibility of:

(VAR x x')
(CONDITION  x -> x', isEven(x)  )

and

(VAR x x')
(CONDITION  x -> x', isEven(s(x))  )

```

The proof of strong joinability of CCPs (10) and (11) above has benefitted from the following observation, which follows from [3, Section 3]: the (in)feasibility of an  $f$ -goal  $F$  with respect to a theory  $\text{Th}$  consisting of implicative sentences  $(\forall \vec{x}) B_1 \wedge B_n \Rightarrow B$  (or definite Horn clauses  $B \leftarrow B_1, \dots, B_n$ ) only depends on the sentences in  $\text{Th}$  defining predicates  $P$  occurring in  $F$  as atoms  $P(t_1, \dots, t_k)$ , i.e., such that  $P(t_1, \dots, t_k) = \sigma(B)$  for some substitution  $\sigma$  and also (recursively) on sentences in  $\text{Th}$  defining predicates in  $B_1, \dots, B_n$ . Then, we collect such sentences as a subtheory  $\text{Th}_{\downarrow F}$  of  $\text{Th}$  and equivalently solve the (in)feasibility problem of  $F$  with respect to  $\text{Th}_{\downarrow F}$ . This has been implemented in the current version of `infChecker` and advantageously used to check strong joinability of (10) and (11) within a 60s. timeout.

## 4 Conclusions

We have improved our tool `infChecker` to give full support to the general notion of feasibility developed in [3]. We have extended the COPS format for CTRSs with new sections to introduce first-order theories  $\text{Th}$  with respect to which a sequence of atoms in the usual section `CONDITION` is checked to prove or disprove feasibility. In particular, we can use `infChecker` now to give support to the analysis of confluence and termination of GTRSs. Regarding *future work*, we plan to extend our confluence and termination tools `CONFident` [5] and `MU-TERM` [4], which use `infChecker` as an auxiliary tool, to improve its ability to prove and disprove confluence and termination of GTRSs. For instance, as shown in Example 4, with the new version of `infChecker` we could obtain a proof of confluence of  $\mathcal{R}$  in Example 1 by using an improved version of `CONFident`. We also plan to handle the new ARI format of CoCo.



## References

- [1] Jürgen Avenhaus and Carlos Loria-Sáenz. On Conditional Rewrite Systems with Extra Variables and Deterministic Logic Programs. In Frank Pfenning, editor, *Logic Programming and Automated Reasoning, 5th International Conference, LPAR'94, Proceedings*, volume 822 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 1994.
- [2] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
- [3] Raúl Gutiérrez and Salvador Lucas. Automatically Proving and Disproving Feasibility Conditions. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 416–435. Springer, 2020.
- [4] Raúl Gutiérrez and Salvador Lucas. MU-TERM: Verify Termination Properties Automatically (System Description). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 436–447. Springer, 2020.
- [5] Raúl Gutiérrez, Salvador Lucas, and Miguel Vitores. Proving Confluence in the Confluence Framework with CONFident. *Fundam. Informaticae*, 192(2):167–217, 2024.
- [6] Salvador Lucas. Context-sensitive Rewriting. *ACM Comput. Surv.*, 53(4):78:1–78:36, 2020.
- [7] Salvador Lucas. Local confluence of conditional and generalized term rewriting systems. *Journal of Logical and Algebraic Methods in Programming*, 136:paper 100926, pages 1–23, 2024.
- [8] Salvador Lucas. Confluence of Almost Parallel-Closed Generalized Term Rewriting Systems. In Clark Barrett and Uwe Waldmann, editors, *Automated Deduction - CADE 30 - 30th International Conference on Automated Deduction, Proceedings*, volume 15493 of *Lecture Notes in Artificial Intelligence*, pages 187–206. Springer Nature Switzerland, 2025.
- [9] Salvador Lucas and Raúl Gutiérrez. Use of logical models for proving infeasibility in term rewriting. *Inf. Process. Lett.*, 136:90–95, 2018.
- [10] Florian Meßner and Christian Sternagel. nonreach - A tool for nonreachability analysis. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *Lecture Notes in Computer Science*, pages 337–343. Springer, 2019.
- [11] A. Middeldorp, J. Nagele, and K. Shintani. Confluence Competition 2019. In D. Beyer, M. Huisman, F. Kordon, and B. Steffen, editors, *Proc. of Tools and Algorithms for the Construction and Analysis of Systems, TACAS'19*, pages 25–40. Springer, 2019.
- [12] Naoki Nishida, Takayuki Kuroda, Makishi Yanagisawa, and Karl Gmeiner. CO3: a CONverter for proving CONfluence of CONditional TRSs. In *4th International Workshop on Confluence, IWC 2015*, page 42, 2015.
- [13] Enno Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002.
- [14] C. Sternagel. CoCo 2020 Participant: ConCon 1.10. In M. Ayala-Rincón and S. Mirmram, editors, *Proc. of the 9th International Workshop on Confluence, IWC'20*, page 65, 2020.
- [15] Sarah Winkler and Georg Moser. Mædmax: A maximal ordered completion tool. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *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 *Lecture Notes in Computer Science*, pages 472–480. Springer, 2018.


# Redeeming Newman; orthogonality in rewriting

## Past, present and future in a 1-algebraic setting

Vincent van Oostrom

University of Sussex, Brighton, United Kingdom  
vvo@sussex.ac.uk

### Abstract

Despite sixty percent of Newman’s seminal 1942 paper being devoted to residual theory, that remains obscure due to that his instantiation of the theory there to the (non-erasing)  $\lambda\beta$ -calculus was fatally flawed. We redeem the approach showing: 1) any rewrite system instantiating his theory induces a so-called 1-ra, an axiomatically orthogonal rewrite system, entailing co-initial reductions have *least* upperbounds; 2) the rewrite system underlying any (non-erasing) syntactically orthogonal TRS instantiates his theory. CC by 4.0 .

**Rewriting** The primary notion in rewriting is from [17]<sup>1</sup>: a *rewrite system*  $\rightarrow := \langle \mathcal{O}, \mathcal{S}, \text{src}, \text{tgt} \rangle$  comprises objects  $\mathcal{O}$  and steps  $\mathcal{S}$  with source, target maps  $\text{src}, \text{tgt}$  from the latter to the former [26, Def. 8.2.2]. Steps  $\phi, \psi$  are *co-initial* if  $\text{src}(\phi) = \text{src}(\psi)$ , *co-final* if  $\text{tgt}(\phi) = \text{tgt}(\psi)$  and *parallel* to each other if both. A *morphism* from  $\rightarrow$  to  $\rightarrow' := \langle \mathcal{O}', \mathcal{S}', \text{src}', \text{tgt}' \rangle$  preserves structure; it maps  $\phi \in \mathcal{S}$  with source  $a$  and target  $b$ , denoted by  $\phi : a \rightarrow b$ , to  $\phi' : a' \rightarrow b'$  in  $\mathcal{S}'$ . Rewrite properties [2, 26] pertain to various rewrite systems *constructed from*  $\rightarrow$ . *E.g.*, the *Church–Rosser* property [5, 17] expresses that for any *conversion* there exists a valley of co-final *reductions* parallel to it, with (finite) reductions and conversions rewrite systems *constructed from*  $\rightarrow$ . As constructions here we will use the 1-operations of *loop* 1, *composition*  $\cdot$ , *reverse*  $^{-1}$  and *residuation*  $/$ , where by a 1-operation we mean an operation respecting sources and

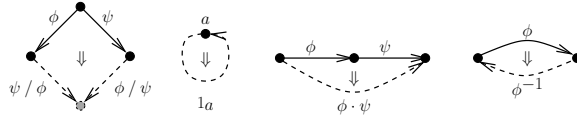


Figure 1: Step-forming operations: *residuation*  $/$ , *loop* 1, *composition*  $\cdot$ , *reverse*  $^{-1}$

targets as depicted by their *generalised* arities [22, Ex. 5.3] in Fig. 1. *E.g.*, composition has two consecutive steps as input (the *full* arrows) and a single step as output (the *dashed* arrow), parallel to each other as depicted. That is, for each 1-operation its *input* arity is the universally quantified (full) subsystem, and its *output* arity is the existentially quantified (dashed) subsystem. The steps in the input arity of composition being *consecutive* captures that composition is only defined on consecutive steps, i.e. if  $\text{tgt}(\phi) = \text{src}(\psi)$  for steps  $\phi, \psi$ . Similarly, residuation  $/$  requires *co-initial* steps in its input.

**1-algebras** To algebraically deal with 1-operations requires to enrich universal algebra. Whereas 1,  $\cdot$  and  $^{-1}$  and laws for them (see Def. 1) are known to be covered by *essentially algebraic theories* [20, Ex. 4], the generalisations needed to smoothly deal with  $/$  are *in statu nascendi* [22]. Acknowledging this, we refer to algebras having a rewrite system  $\rightarrow$  as carrier and 1-operations among those in Fig. 1 as *1-algebras*. This allows to reuse, as we do, terminology from algebra.

<sup>1</sup>In Newman’s combinatorial topology-inspired words: *We are concerned with two kinds of entities, “objects” and the “moves” performed on them, and each move is associated with two objects, “initial” and “final.”*

$l\text{-unit}(\varrho)$	:	$1 \cdot \varrho \Rightarrow \varrho$	$invol\text{-}id$	:	$1^{-1} \Rightarrow 1$
$r\text{-unit}(\varrho)$	:	$\varrho \cdot 1 \Rightarrow \varrho$	$anti\text{-}auto(\varrho, \varsigma)$	:	$(\varrho \cdot \varsigma)^{-1} \Rightarrow \varsigma^{-1} \cdot \varrho^{-1}$
$assoc(\varrho, \varsigma, \zeta)$	:	$(\varrho \cdot \varsigma) \cdot \zeta \Rightarrow \varrho \cdot (\varsigma \cdot \zeta)$	$invol(\varrho)$	:	$(\varrho^{-1})^{-1} \Rightarrow \varrho$

Table 1: 1-algebra laws / proofterm rewrite rules

**Definition 1.** A 1-*monoid* is a 1-algebra with 1-operations 1 and  $\cdot$  satisfying the (pertaining) laws in Tab. 1.<sup>2</sup> We use  $\Rightarrow$  to denote (the carrier of) the *free* 1-monoid induced by  $\rightarrow$  and refer to its elements as *reductions* [26, Def. 8.2.10]. A 1-*involutive* 1-monoid is a 1-algebra with 1-operations 1,  $\cdot$  and  $^{-1}$  satisfying the laws in Tab. 1.<sup>2</sup> We use  $\rightsquigarrow$  to denote (the carrier of) the *free* 1-involutive 1-monoid, cf. [7], induced by  $\rightarrow$  and refer to its elements as *conversions*.

**Example 1.** Any algebra can be viewed as a 1-algebra by viewing its carrier as a single-object rewrite system having a step on it for each element of the algebra. Accordingly, algebra examples of 1-monoids are  $\langle \mathbb{Z}, 0, + \rangle$  and  $\langle \mathbb{N}, 0, + \rangle$ , and algebra examples of 1-involutive 1-monoids are  $\langle \mathbb{Z}, 0, +, (-) \rangle$  and  $\langle \mathbb{N}, 0, +, id \rangle$ . A 1-algebra example is (finite) walks in a graph with operations *empty*, *composition*, and *reverse*, or more generally paths in space.

Freeness of  $\Rightarrow$  means that any morphism from  $\rightarrow$  to a 1-monoid factors into a morphism from  $\rightarrow$  to  $\Rightarrow$  and (a 1-monoid preserving) one from  $\Rightarrow$  to the 1-monoid, and similarly for  $\rightsquigarrow$ .

**Example 2.** The *length*-morphism maps each step  $\phi$  to 1 in the involutive monoid  $\langle \mathbb{N}, 0, +, id \rangle$ . It factors through mapping the *step*  $\phi$  to the *conversion*  $\phi$ , and so does the *relation*-morphism mapping  $\phi$  to  $(src(\phi), tgt(\phi))$  in the *equivalence* closure, *convertibility*, of the rewrite relation of  $\rightarrow$ , equipped with the expected operations. Similarly, that same morphism into the *reflexive-transitive* closure, *reducibility*, factors through mapping the *step*  $\phi$  to the *reduction*  $\phi$ .

(1-involutive) 1-monoids are (dagger) categories, and them being essentially algebraic means that free such can be defined syntactically [20]: letting the operations 1,  $\cdot$  and  $^{-1}$  double as *function symbols* of arities 0, 2 and 1, one can inductively build terms from the steps of  $\rightarrow$  respecting sources and targets. Such terms we refer (red) to as *proofterms* [26, Ch. 8] as they are *terms* that can be conceived of as *proofs* (of *reducibility* of their source and target in case of  $\Rightarrow$  and of their *convertibility* in case of  $\rightsquigarrow$ ) in (sub-)equational logic(s) [18] induced by  $\rightarrow$ . To quotient out the (pertaining) laws from proofterms, one may use (proof)term rewriting itself: orienting the laws into rules on proofterms as in Tab. 1 yields a complete (confluent and terminating) (proof)term rewrite system  $\Rightarrow$ , with a  $\Rightarrow$ -normal form being either a single 1 or a right-branching  $\cdot$ -tree of (reversed)  $\rightarrow$ -steps, *i.e.*  $\Rightarrow$ -normal forms are in 1-1 correspondence with the usual notion of a reduction (conversion) as sequence of forward (and backward)  $\rightarrow$ -steps [5, 26]. This then allows to *define* the 1-operations 1,  $\cdot$  (and  $^{-1}$ ) on reductions (and conversions) as the proofterm-forming operation of applying the corresponding function symbol followed by  $\Rightarrow$ -normalisation [7, App. A].

**Example 3.** As running example we employ Kleene's rewrite system  $\rightarrow$  [26, Fig. 1.2] comprising the four steps  $\phi: a \rightarrow a'$ ,  $\phi': a' \rightarrow a$ ,  $\psi: a \rightarrow b$ , and  $\chi: a' \rightarrow c$ . Then  $\phi \cdot \psi$  is not a proofterm since the target  $a'$  of its 1<sup>st</sup> step  $\phi$  is distinct from the source  $a$  of its 2<sup>nd</sup> step  $\psi$ . Among proofterms  $\varrho := (\phi'^{-1} \cdot \phi^{-1})^{-1}$ ,  $\varrho' := \phi \cdot \phi'$ ,  $\varsigma := (\phi' \cdot \phi) \cdot \phi^{-1}$  and  $\varsigma' := \phi' \cdot (\phi \cdot \phi^{-1})$ , both  $\varrho'$  and  $\varsigma'$  are conversions, but only the former is a reduction. Because both  $\varrho$  and  $\varsigma$  are  $\Rightarrow$ -reducible neither is a conversion; their  $\Rightarrow$ -normal forms are,  $\varrho'$  and  $\varsigma'$ .

<sup>2</sup> For the moment reading the proofterm rewrite rules as laws (symmetrically) for the 1-operations. We have left the assumptions [20, Ex. 4] on sources and targets of  $\varrho, \varsigma, \zeta$  and 1 implicit, to stress similarity with algebra.

**Orthogonality** We recast the diamond and cube properties, cf. [26, Sec. 8.7], 1-algebraically.

**Definition 2.**  $\rightarrow$  has the *diamond* property (DP), if for all co-initial  $\phi, \psi$ , there exist co-final  $\psi', \phi'$  such that  $\phi \diamond \psi$ , where  $\phi \diamond \psi$  denotes that  $\phi, \psi, \psi', \phi'$  constitute a *diamond*:  $\text{src}(\phi) = \text{src}(\psi) \& \text{tgt}(\phi) = \text{src}(\psi') \& \text{tgt}(\psi) = \text{src}(\phi') \& \text{tgt}(\psi') = \text{tgt}(\phi')$ .  $\rightarrow$  is *confluent* if  $\rightarrow$  has the DP.

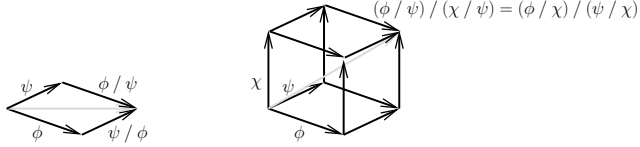


Figure 2: The diamond property ( $\diamond$ ; left) vs. the cube property ( $\boxplus$ ; right)

Using that a *peak* (*valley*) [5] is a conversion of shape  $\leftarrow \cdot \rightarrow$  ( $\rightarrow \cdot \leftarrow$ ), DP expresses that for every peak there exists a valley parallel to it. Using a skolem-function  $/$  (*residuation* in Fig. 1) to witness  $\psi'$  by  $\psi / \phi$  and  $\phi'$  by  $\phi / \psi$ ,<sup>3</sup> DP can be expressed 1-algebraically as: for all co-initial  $\phi, \psi$ ,  $\psi / \phi \diamond \phi / \psi$ . Thus,  $\rightarrow$  has the DP iff  $\langle \rightarrow, / \rangle$  is a 1-algebra for *some* residuation  $/$ .

**Example 4.** In Ex. 3, for the  $\rightarrow$ -peak  $\psi^{-1} \cdot \phi$  the  $\rightarrow$ -valley  $\psi^{-1} \cdot \phi'^{-1}$  is parallel to it, but  $\rightarrow$  is not confluent since for the peak  $\psi^{-1} \cdot (\phi \cdot \chi)$  there is no valley ( $b$  and  $c$  are  $\rightarrow$ -normal forms).

In [17, Sec. 1], Newman explained confluence of a rewrite system  $\rightarrow$  in terms of its reducibility quasi-order  $\rightarrow$  having *upperbounds*. He left the determination of conditions required for having *least* upperbounds for later. We put forward such conditions in [26, Sec. 8.7] in the form of the laws on residuation in Tab. 2, proposing to call any rewrite system satisfying (1)–(4) *orthogonal*, cf. [15, 9] (Fig. 2 depicts law (4) going back to [17, Thm. 5( $\Delta_4$ )] dubbed *cube* in [13, Lem. 2.2.1]). Here we recast that account 1-algebraically to then instantiate it in the next sections.

$\phi / 1 = \phi$	(1)		
$\phi / \phi = 1$	(2)	$\chi / (\phi \cdot \psi) = (\chi / \phi) / \psi$	(5)
$1 / \phi = 1$	(3)	$(\phi \cdot \psi) / \chi = (\phi / \chi) \cdot (\psi / (\chi / \phi))$	(6)
$(\phi / \psi) / (\chi / \psi) = (\phi / \chi) / (\psi / \chi)$	(4)	$1 \cdot 1 = 1$	(7)

Table 2: Laws of a 1-ra (left) and of a 1-rac (also right)

**Definition 3.** A *1-residual algebra* (1-ra) is a 1-algebra  $\langle \rightarrow, 1, / \rangle$  such that (1)–(4) in Tab. 2 hold. A *1-rac* (1-ra with *composition*) is a 1-algebra  $\langle \rightarrow, 1, /, \cdot \rangle$  such that (1)–(7) hold.

**Example 5.**  $\langle \mathbb{N}, 0, -, + \rangle$  is a 1-rac, so  $\langle \mathbb{N}, 0, - \rangle$  is a 1-ra, for  $-$  *monus* (cut-off subtraction).

In a 1-ra(c) there is a *natural* order on co-initial steps given by  $\phi \preceq \psi := (\phi / \psi = 1)$ . Quotienting out  $\preceq \cap \succeq$  yields a 1-ra(c) again whose natural order is a partial order [26, Lem. 8.7.25(iii) and 8.7.41(ii)]. Key to referring to a rewrite system  $\rightarrow$  constituting a 1-ra as being *orthogonal*, is that any such induces a 1-rac on  $\rightarrow$  [26, Lem. 8.7.47], which then has *least* upperbounds [26, Exc. 8.7.40(ii)]. This is characterised, using categorical language to be (ex/comp)act, by:

**Theorem 1** (cf. [25, 15, 21]).  $\langle \rightarrow, 1, /, \cdot \rangle$  is a 1-rac whose natural order is a partial order, where  $\phi / \psi := \phi'$  for every peak  $\phi, \psi$  and its pushout valley  $\psi', \phi'$  (in the categorical sense) iff  $\langle \rightarrow, 1, \cdot \rangle$  is a 1-monoid that is left-cancellative (each  $\chi$  is epi: for all  $\phi, \psi$ , if  $\chi \cdot \phi = \chi \cdot \psi$  then  $\phi = \psi$ ), gaunt (isomorphisms are 1) and has pushouts; i.e. *lubs of peaks exist*.

<sup>3</sup>We *a priori* get 2 skolem-functions,  $f(\phi, \psi)$  for  $\psi'$  and  $g(\phi, \psi)$  for  $\phi'$ , but may assume  $f(\phi, \psi) = g(\psi, \phi)$ .

**Redeeming Newman** In 1942 in [17], Newman refactored the proof of the Church–Rosser property for the  $\lambda I$ -calculus in [5], by abstracting from the  $\lambda$ -term-structure of objects, factoring the proof through an *axiomatisation* of a function  $|$  entailing confluence [17, Sec. 8–12], and showing the axioms to be satisfied for the  $\lambda I$ -calculus [17, Sec. 13,14]. The latter was later found to be erroneous [23] due to confusing variables when working with  $\lambda$ -terms modulo  $\alpha$ -equivalence.<sup>4</sup> We redeem his approach, showing in this section his main result [17, Thm. 5] factors through orthogonality, and in the next that it applies to (non-erasing) OTRS. To present his result,<sup>5</sup> let  $|$  yield for co-initial  $\phi, \psi$  a (finite) set  $\phi|\psi$  of  $\rightarrow$ -steps from  $\text{tgt}(\psi)$ , the  $\psi$ -derivates of  $\phi$ ; it lifts to (finite) sets  $\Phi$  of steps by  $\Phi|\psi := \bigcup_{\phi \in \Phi} \phi|\psi$  and to reductions by  $\Phi|(\varrho \cdot \psi) := (\Phi|\varrho)|\psi$  and  $\Phi|1 := \Phi$ . A *development* of  $\Phi$  is a  $\rightarrow$ -reduction in which only derivates of steps in  $\Phi$  occur and no remain [5, 3, 26]. Using Newman’s notions and our notations, his result reads:

**Theorem 2** ([17]). *For co-initial reductions  $\varrho, \varsigma$  and set of steps  $\Phi$ , there are reductions  $\varsigma', \varrho'$  such that  $\varrho \circ_{\varrho'} \varsigma'$  and  $\Phi|(\varrho \cdot \varsigma') = \Phi|(\varsigma \cdot \varrho')$  if axioms  $\Delta_1$ – $\Delta_4$  hold and  $J_1, J_2$  for a predicate  $J$ :*

- ( $\Delta_1$ )  $\phi|\psi = \emptyset$  iff  $\phi = \psi$ ;
- ( $\Delta_2$ ) if  $\phi \neq \psi$ , then  $(\phi|\chi) \cap (\psi|\chi) = \emptyset$ ;
- ( $\Delta_3$ ) if  $\phi \neq \psi$ , then there exist co-final developments  $\varrho$  of  $\psi|\phi$ , and  $\varsigma$  of  $\phi|\psi$ ;
- ( $\Delta_4$ ) for  $\varrho$  and  $\varsigma$  in ( $\Delta_3$ ),  $\chi|(\phi \cdot \varrho) = \chi|(\psi \cdot \varsigma)$ ;
- ( $J_1$ ) If  $\phi J \psi$ , then  $\phi|\psi$  has precisely one member;
- ( $J_2$ ) If  $\psi_1 \in \phi_1|\chi$  and  $\psi_2 \in \phi_2|\chi$ , and if  $\phi_1 J \phi_2$  or  $\phi_1 = \phi_2$ , then  $\psi_1 J \psi_2$  or  $\psi_1 = \psi_2$ .

Let the *parallel* rewrite system  $\multimap$  have as objects the objects of  $\rightarrow$ , and a step  $\Phi_a$  if  $\Phi$  is a set of steps *at*  $a$  that is a  $J$ -set [17, p. 232]: (distinct) steps in  $\Phi$  are from  $a$  and pairwise  $J$ -related. Then the source of  $\Phi_a$  is  $a$  and its target is the target of a development of  $\Phi$ .

**Lemma 1.** *Under the assumptions of Thm. 2,  $\langle \multimap, \emptyset, | \rangle$  is a 1-ra, so  $\multimap$  is orthogonal.*

*Proof.* That  $\multimap$  is well-defined, i.e. that  $\Phi_a$  has a unique target holds by [17, Lem. 2]. [17, Lem. 5.1] shows both that  $|$  is a residuation (Fig. 1) for  $\multimap$ , so  $\multimap$  has the DP, and that it has the cube property, i.e. law (4) holds. Laws (1)–(3) are seen to hold by easy inductions.  $\square$

*Proof of Thm. 2.* As  $\rightarrow$ -reductions are (singleton)  $\multimap$ -reductions and the 1-ra on  $\multimap$  induces a 1-rac on  $\multimap$  by the previous section,<sup>6</sup> one concludes by setting  $\varsigma' := \varsigma|\varrho$  and  $\varrho' := \varrho|\varsigma$ .  $\square$

**Steps-as-terms** To apply Thm. 2 to a *term* rewrite system [2, 26]  $\mathcal{T} := \langle \Sigma, P \rangle$  given by a *signature*  $\Sigma$  and a set  $P$  of *rules*  $\rho: \ell \rightarrow r$  for  $\ell, r$  terms over function symbols in  $\Sigma$  (and variables), we let its *multistep* rewrite system  $\multimap_{\mathcal{T}}$  have as objects terms over  $\Sigma$  and *steps-as-terms*, i.e. as steps terms over  $\Sigma \sqcup P$ , with  $\text{src}$  ( $\text{tgt}$ ) the homomorphic extension of the function mapping rule symbols to their left-(right-)hand side. The *parallel* rewrite system  $\multimap_{\mathcal{T}}$  and (single step) rewrite system  $\rightarrow_{\mathcal{T}}$  arise from  $\multimap_{\mathcal{T}}$  by restricting rule-symbols to occur *at parallel positions* respectively *once* in steps [26, Prop. 8.2.22]. We employ that  $\multimap_{\mathcal{T}}$  is orthogonal (in the above sense) if  $\mathcal{T}$  is an OTRS, a TRS that has left-linear and non-overlapping rules [2, 26], i.e. that the residuation / given in [26, Def. 8.7.4] induces a 1-ra on  $\multimap_{\mathcal{T}}$  [26, Prop. 8.7.7(ii)] for OTRS.

**Lemma 2.** *For an OTRS  $\mathcal{T}$ , the assumptions of Thm. 2 hold for  $\rightarrow_{\mathcal{T}}$  when:*

- defining  $\phi J \psi$  if  $\phi, \psi$  are steps (whose rule symbols are) at parallel positions; and

<sup>4</sup>It being the first of its kind, the error could be called the  $\alpha$ - $\alpha$ -error (lucerne-error?).

<sup>5</sup>See [17] for more. We present Newman’s axioms and result as is, as our only goal is to *instantiate* them.

<sup>6</sup>Alternatively, this can be concluded from that the assumptions of Thm. 2 entail properties 1–4 of [15].

- letting  $\phi \mid \psi$  be the set of steps occurring in (the parallel step)  $\phi / \psi$ .

*Proof.* That the axioms hold follows from well-known facts for parallel steps in OTRS [2, 26], e.g.,  $J_2$  corresponds to the *Disjointness Property* in [26, Ex. 8.6.30] and  $\Delta_4$  to *cube* [13, Lem. 2.2.1], **permutation** [15]. (Using steps-as-terms, the axioms can also be verified by *inductions* on steps.) Note that non-erasingness is (only) needed for  $\Delta_1$ , cf. [26, Ex. 8.7.24].  $\square$

**Conclusions** We ruminate about the past, present and future of orthogonality.

**Past (the curious case of orthogonality in rewriting)** It is curious that Newman starts out [17] with stating to leave the study of *least* upperbounds (lubs) for later, to then devote most of the paper to introducing conditions that guarantee the very existence of lubs, as shown above (Lem. 1). That reductions constitute a lattice (have lubs) was shown only much later (in the 70s) for concrete rewrite systems such as recursive programs and the  $\lambda\beta$ -calculus, see [13], subsequently axiomatised and couched in categorical language (existence of pushouts) in [25, 15, 21]. Here we cast our account of that [26, Sec. 8.7] 1-algebraically (Def. 3 and Thm. 1).

$$\beta(x, y, z) : B x y z \rightarrow x (y z) \quad \gamma(x, y, z) : C x y z \rightarrow x z y \quad \iota(x) : I x \rightarrow x$$

Table 3: Combinatory Logic: term rewrite rules  $P$  of  $\mathcal{BCI}$  in applicative notation

**Example 6** (illustrating that lubs are subtle to define).  $\mathcal{BCI}$  is the TRS with signature  $\Sigma := \{B, C, I, @\}$  and rules  $P := \{\beta, \gamma, \iota\}$  (Tab. 3). Since the rules of the Combinatory Logic  $\mathcal{BCI}$  in Tab. 3 are left-linear and non-overlapping,  $\dashv\vdash_{\mathcal{BCI}}$  is orthogonal by the above. Consider the peak  $I x \xrightarrow{\phi} I(I x) \xrightarrow{\psi} I x$  for the steps  $\phi := \iota(I x)$  and  $\psi := I \iota(x)$ . (Note that  $\phi, \psi$  are extensionally the same but not intensionally so.) The lub (pushout) of the peak is formed *not* by the empty valley from  $I x$ , but by the valley comprising twice the step  $\iota(x) : I x \rightarrow_{\mathcal{BCI}} x$  since  $\psi / \phi = \iota(x) = \phi / \psi$ . That is, that  $\phi, \psi$  are intensionally different, perform different work, constitute a *syntactic accident* [13, p. 34], is reflected in their pushout (lub) not being empty.

Despite the prominence of Combinatory Logic and the  $\lambda\beta$ -calculus since the 30s, it took until the 80s to clepe them *orthogonal* term rewrite systems, making them in retrospect the 1<sup>st</sup> and 2<sup>nd</sup> (1<sup>st</sup> and 2<sup>nd</sup>-order) such. But that definition of orthogonality is *syntactic*, asks rule(s) to be left-linear and non-overlapping [2, 26], pertains to *terms* only. That led to the second curiosity that on the one hand many structured rewrite systems having lubs, e.g., *interaction nets* [12], *braids* [16], *self-distributivity* [24], ... were *not* covered by that syntactic definition, and that on the other hand that syntactic definition was found to be *lacking*, to not guarantee confluence let alone existence of lubs, already for minor generalisations of term rewriting:

**Example 7.** The rules  $a \rightarrow b$  and  $f(x) \rightarrow c \Leftarrow x = a$  are left-linear and non-overlapping but not even confluent as witnessed by both  $f(b)$  and  $c$  being normal forms in the peak  $f(b) \leftarrow f(a) \rightarrow c$ .

Whence we propose(d [26, Sec. 8.9]) to factor the syntactic definition of orthogonality through its associated rewrite system being orthogonal, to constitute a 1-ra (Def. 3), making it interesting to see whether for systems in the literature the former entails the latter. For orthogonal TRS and PRS this is known to be the case [26, Sec. 8.7][4] by associating  $\dashv\vdash_{\mathcal{T}}$  to  $\mathcal{T}$  with the proof unifying [5] with *axiomatic* [17] and *inductive* [3, Sec. 3.2] (TML) proofs. By *steps-as-terms* being conceptually parsimonious the residuation-based proof is superior to TML, which is based on *ad hoc proof trees*, as we illustrate for  $\mathcal{BCI}$  in Ex. 8. Ex. 9 exemplifies that for certain *context-sensitive* and *conditional* TRS, orthogonality indeed induces orthogonality. We leave it to future research to check other syntactically orthogonal systems in the literature.



**Example 8** (orthogonality of  $\multimap_{\mathcal{BCI}}$  for OTRS  $\mathcal{BCI}$ ). The (proof)terms  $\phi := B I \gamma(C, x, y) \iota(z)$  and  $\psi := \beta(I, C C x y, I z)$  yield a peak  $B I (C y x) z \leftarrow_{\phi} B I (C C x y) (I z) \rightarrow_{\psi} I (C C x y) (I z)$  for which residuation (per [26, Def. 8.7.4]) yields the valley  $B I (C y x) z \rightarrow_{\psi'} I (C y x z) \leftarrow_{\phi'} I (C C x y (I z))$ , the lub (pushout) of  $\phi, \psi$ , formed by (proof)terms  $\psi' := \psi / \phi := \beta(I, C y x, z)$  and  $\phi' := \phi / \psi := I(\gamma(C, x, y) \iota(z))$ , as one may check in Haskell or ProTeM.

**Example 9.**  $\langle \multimap, 1, / \rangle$  is a 1-ra for  $/$  as in Ex. 8 [26, Def. 8.7.4], if steps  $\phi$  are restricted by:<sup>7</sup>

- (i) for CSR as in [14, Thm. 8.12]: all frozen arguments are  $\Sigma$ -terms, or
- (ii) for orthogonal normal CTRS [26, Sec. 4.11.2]: if  $\phi = \rho(\vec{\phi})$  for rule  $\rho(\vec{x}) : \ell \rightarrow r \leftarrow \overline{\ell} \rightarrow r$ , then for all  $j$ ,  $\ell_j^{\sigma} \rightarrow r_j^{\sigma}$  (observe  $r_j^{\sigma} = r_j$  by normality), where  $\sigma(x_i) := \text{src}(\phi_i)$  for all  $i$ .

**Present (inappropriate appropriation)** Rewrite systems being basic small wonder they occur elsewhere nowadays, *e.g.*, as *multidigraphs*, *quivers* in representation theory [8], *pre-categories* in Garside theory [6] or 1-*polygraphs* in higher-dimensional group presentations [1]. As much as we would like to base ourselves on [6, 1], we cannot as both accounts are inadequate for our key notions *conversion* and *residuation*. More generally, *subsystems* [18] ( $\leftrightarrow$  and  $\rightarrow^+$ , the free 1-algebras with  $^{-1}$  respectively  $\cdot$  satisfying the laws in Tab. 1) and *supersystems* [26] (infinite(*e*/ary) reductions) are absent from them, and so is (must be) classical rewrite theory.

$$\begin{array}{llll} l\text{-inv}(\varrho) & : & \varrho^{-1} \cdot \varrho & \Rightarrow 1 & l\text{-inv-}x(\varrho, \varsigma) & : & \varrho^{-1} \cdot (\varrho \cdot \varsigma) & \Rightarrow \varsigma \\ r\text{-inv}(\varrho) & : & \varrho \cdot \varrho^{-1} & \Rightarrow 1 & r\text{-inv-}x(\varrho, \varsigma) & : & \varrho \cdot (\varrho^{-1} \cdot \varsigma) & \Rightarrow \varsigma \end{array}$$

Table 4: 1-algebra laws / proofterm rewrite rules for 1-groups, extending Tab. 1

**Remark.** Modelling conversions as  $\leftrightarrow$ -reductions is *too weak*, so 1-polygraphs are [11, Sec. 2.4], as *categories* (1-monoids) miss out on involution [7],<sup>8</sup> and assuming cancellation is *too strong* [17, Sec. 1], as *groupoids* (1-groups; see Tab. 4<sup>9</sup>) lose embedding [19]. Indeed, [6, 1] have algebraic accounts of neither conversion nor residuation, so cannot account for orthogonality, 1-ra's (having composition without residuation is analogous to having addition without monus; we do not know of other accounts where both are treated on a par, algebraically, as we think they should).

**Future (explorations)** (I) We restricted attention to the axioms in [17] for the *non-erasing*  $\lambda\beta$ -calculus; ( $\Delta_1$ ). These entail additional properties, *e.g.*, the natural order on  $\multimap$ -steps is the *subset* order, all developments have the same length [19], and  $\rightarrow$  is *normalising* (WN) iff it is *terminating* (SN) [17, Thm. 8][26, Thm. 4.8.5]. Which are retained for the axioms in [17] for the (*erasing*)  $\lambda\beta$ -calculus? orthogonality? (II) Does meta-theory such as that the *full*  $\multimap$ -strategy is (hyper-)normalising, generalise? (III) Does viewing a *proof order* (see [26, Thm. 7.5.12]) as a morphism from conversions into a (well-founded) 1-involutive 1-monoid have advantages beyond those in [7]? (IV) Does the approach generalise to *infinite(e/ary)* reductions? (V) Can the *Grothendieck group construction* be based on orthogonality? (VI) How to *formalise* this algebraic approach, in particular *residuation*, *cf.* [10]? (VII) How to generalise proofs via steps-as-terms to other structures, *e.g.*, to *steps-as-port-graphs* for interaction nets [12].

**Acknowledgments** We thank Nao Hirokawa, Vít Jelínek, Philip Saville and Fer-Jan de Vries for discussions and feedback on a previous version, and the IWC reviewers for reviews.

<sup>7</sup>As before<sup>5</sup>, we refer the reader to the cited literature for more, for reasons of room.

<sup>8</sup>Involution is *essential* and *useful*; it saved half the work in formalising [7] (B. Felgenhauer; pers. comm.).

<sup>9</sup>The proofterm rewrite rules are obtained by 1-completion, see Ch. 7 of either [2, 26]: though *l-inv-x* and *r-inv-x* are *derivable* they need to be adjoined to turn  $\Rightarrow$  into a *complete* proofterm rewrite system.

## References

- [1] D. Ara, A. Burroni, Y. Guiraud, P. Malbos, F. Métayer, and S. Mimram. Polygraphs: From rewriting to higher categories, 2023. doi:10.48550/arXiv.2312.00429.
- [2] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
- [3] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 1984.
- [4] H.J.S. Bruggink. *Equivalence of Reductions in Higher-Order Rewriting*. PhD thesis, Utrecht University, 2008. URL: <https://dspace.library.uu.nl/handle/1874/27575>.
- [5] A. Church and J.B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39:472–482, 1936. doi:10.2307/1989762.
- [6] P. Dehornoy, F. Digne, E. Godelle, D. Krammer, and J. Michel. *Foundations of Garside Theory*. European Mathematical Society, 2015. doi:10.4171/139.
- [7] B. Felgenhauer and V. van Oostrom. Proof orders for decreasing diagrams. In *RTA 2013*, volume 21 of *LIPICs*, pages 174–189, 2013. doi:10.4230/LIPICs.RTA.2013.174.
- [8] P. Gabriel. Unzerlegbare Darstellungen I. *Manuscripta Math.*, 6(1):71–103, 1972. doi:10.1007/BF01298413.
- [9] J.R.W. Glauert and Z. Khasidashvili. Relative normalization in deterministic residual structures. In *CAAP’96*, volume 1059 of *LNCS*, pages 180–195, 1996. doi:10.1007/3-540-61064-2\_37.
- [10] C. Kirk. Proof terms for term rewriting. *Archive of Formal Proofs*, 2025. Formal proof development in Isabelle. URL: [https://isa-afp.org/entries/Proof\\_Terms\\_Term\\_Rewriting.html](https://isa-afp.org/entries/Proof_Terms_Term_Rewriting.html).
- [11] N. Kraus and J. von Raumer. A rewriting coherence theorem with applications in homotopy type theory. *MSCS*, 32(7):982–1014, 2022. doi:10.1017/S0960129523000026.
- [12] Y. Lafont. Interaction nets. In *17th POPL*, pages 95–108, 1990. doi:10.1145/96709.96718.
- [13] J.-J. Lévy. *Réductions correctes et optimales dans le  $\lambda$ -calcul*. Thèse de doctorat d’état, Université Paris VII, 1978. URL: <http://pauillac.inria.fr/~levy/pubs/78phd.pdf>.
- [14] S. Lucas. Context-sensitive rewriting. *ACM Comput. Surv.*, 53(4), 2020. doi:10.1145/3397677.
- [15] P.-A. Melliès. Axiomatic rewriting theory VI residual theory revisited. In *RTA 2002*, volume 2378 of *LNCS*, pages 24–50, 2002. doi:10.1007/3-540-45610-4\_4.
- [16] P.-A. Melliès. Braids described as an orthogonal rewriting system, 2009. 22 pp. URL: <https://www.irif.fr/~mellies/papers/braids-for-roel.pdf>.
- [17] M.H.A. Newman. On theories with a combinatorial definition of “equivalence”. *Annals of Mathematics*, 43(2):223–243, 1942. doi:10.2307/1968867.
- [18] V. van Oostrom. Sub-Birkhoff. In *FLOPS 2004*, volume 2998 of *LNCS*, pages 180–195. Springer, 2004. doi:10.1007/978-3-540-24754-8\_14.
- [19] V. van Oostrom and Y. Toyama. Normalisation by random descent. In *FSCD*, volume 52 of *LIPICs*, pages 32:1–32:18, 2016. doi:10.4230/LIPICs.FSCD.2016.32.
- [20] E. Palmgren and S.J. Vickers. Partial horn logic and cartesian categories. *Annals of Pure and Applied Logic*, 145(3):314–353, 2007. doi:10.1016/j.apal.2006.10.001.
- [21] G.D. Plotkin, 1980? Handwritten unpublished notes (communicated by J.W. Klop in Sept. 2022).
- [22] J. Rosický and G. Tendas. Towards enriched universal algebra, 2024. doi:10.48550/arXiv.2310.11972.
- [23] J.B. Rosser. Review of “a new proof of the Church–Rosser theorem”. *JSL*, 21(4):337–426, 1956.
- [24] R.J. Schikora. On orthogonality of self-distributivity. Master’s thesis, University of Innsbruck, 2022. URL: <https://diglib.uibk.ac.at/download/pdf/8160517.pdf>.
- [25] E.W. Stark. Concurrent transition systems. *TCS*, 64:221–269, 1989. doi:10.1016/0304-3975(89)90050-9.
- [26] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.



# Confluence Competition 2025

Rául Gutiérrez<sup>1</sup>, Aart Middeldorp<sup>2,3</sup>, Naoki Nishida<sup>3</sup>,  
Teppei Saito<sup>4</sup>, and René Thiemann<sup>2</sup>

<sup>1</sup> Universitat Politècnica de València, Valencia, Spain

<sup>2</sup> Department of Computer Science, University of Innsbruck, Austria

<sup>3</sup> Department of Computing and Software Systems, Nagoya University, Japan

<sup>4</sup> School of Information Science, JAIST, Japan

The next few pages in these proceedings contain the descriptions of the tools participating in the 14th Confluence Competition (CoCo 2025). CoCo is a yearly competition in which software tools attempt to automatically (dis)prove confluence and related properties of rewrite systems in a variety of formats. For a detailed description we refer to [1]. This year there were 15 tools (listed in order of registration) participating in 5 categories (listed in order of first appearance in CoCo):

	TRS	CTRS	INF	CSR	LCTRS
Grackle-CSI	✓				
crest					✓
Natto			✓		
AProVE	✓				
Hakusan	✓				
CONFident	✓	✓		✓	
infChecker			✓		
CeTA	*		*		
ACP	✓	✓			
CO3		✓	✓		
CRaris					✓
FORT-h	✓				
CSI	✓				
FORTify	*				
SOL				✓	

Since 2024 CoCo adopts the ARI<sup>1</sup> format for input problems. Tools producing certifiable output in a specific category team up with a certifier and participate as combination in that category. The certifiers in CoCo 2025 are CeTA and FORTify. The latter teamed up with FORT-h. The former with ACP, CSI and Hakusan in the TRS category, and with Natto in the INF category.

The winning (for combined YES/NO answers) tools<sup>2</sup> of CoCo 2024 participated as demonstration tools, to provide a benchmark to measure progress. The live run of CoCo 2025 on StarExec [2] Miami can be viewed at

<sup>1</sup><https://project-coco.uibk.ac.at/ARI/>

<sup>2</sup>They are not listed in the table but see <http://project-coco.uibk.ac.at/2024/results.php>.

<https://ari-cops.uibk.ac.at/liveview?comp=CoCo.2025.competition>

Further information about CoCo 2025, including a description of the categories and detailed results, can be obtained from <http://project-coco.uibk.ac.at/2025/>.

**Acknowledgements** The CoCo steering committee is grateful to Geoff Sutcliffe for making StarExec Miami available for CoCo 2025.

## References

- [1] Aart Middeldorp, Julian Nagele, and Kiraku Shintani. CoCo 2019: Report on the Eighth Confluence Competition. *International Journal on Software Tools for Technology Transfer*, 2021. doi: [10.1007/s10009-021-00620-4](https://doi.org/10.1007/s10009-021-00620-4).
- [2] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. StarExec: A Cross-Community Infrastructure for Logic Solving. In *Proc. 7th International Joint Conference on Automated Reasoning*, volume 8562 of *LNCS (LNAI)*, pages 367–373, 2014. doi: [10.1007/978-3-319-08587-6\\_28](https://doi.org/10.1007/978-3-319-08587-6_28).

# CoCo 2025 Participant: CSI-Grackle

Liao Zhang<sup>1,2</sup> Qinxiang Cao<sup>2</sup>

<sup>1</sup> University of Innsbruck, Innsbruck, Tyrol, Austria  
zhangliao714@gmail.com

<sup>2</sup> Shanghai Jiao Tong University, Shanghai, China  
caoqinxiang@sjtu.edu.cn

CSI is an automatic tool for establishing or refuting confluence and related properties of first-order term rewrite systems (TRSs). Development of the tool commenced in 2010. The name “CSI” is derived from the confluence of the Sill and Inn rivers in Innsbruck. The tool is available at: <http://cl-informatik.uibk.ac.at/software/csi>. A detailed description of CSI can be found in [3, 2].

Grackle-CSI [4] builds upon CSI but introduces a new strategy invented by a general-purpose strategy optimizer Grackle [1]. Empirical experiments demonstrate that the invented strategies can (dis)prove more TRSs in ARI-COPS than CSI’s competition strategy for CoCo 2024 while maintaining the same computation budget. The code is available at: <https://github.com/Zhang-Liao/grackle-csi>.

Grackle-CSI will participate in the TRS category of CoCo 2025.

## References

- [1] Mikoláš Janota Jan Hůla, Jan Jakubův and Lukáš Kubej. Targeted configuration of an smt solver. In *International Conference on Intelligent Computer Mathematics*, pages 256–271. Springer, 2022.
- [2] Julian Nagele, Bertram Felgenhauer, and Aart Middeldorp. Csi: New evidence—a progress report. In *International Conference on Automated Deduction*, pages 385–397. Springer, 2017.
- [3] Harald Zankl, Bertram Felgenhauer, and Aart Middeldorp. Csi—a confluence tool. In *Automated Deduction—CADE-23: 23rd International Conference on Automated Deduction, Wrocław, Poland, July 31-August 5, 2011. Proceedings 23*, pages 499–505. Springer, 2011.
- [4] Liao Zhang, Fabian Mitterwallner, Jan Jakubův, and Cezary Kaliszyk. Automated strategy invention for confluence of term rewrite systems. *arXiv preprint arXiv:2411.06409*, 2024.

# CoCo 2025 Participant: **crest** 1.0\*

Jonas Schöpf and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Innsbruck, Austria  
{jonas.schoepf,aart.middeldorp}@uibk.ac.at

The Constrained REwriting Software Tool (**crest**, for short) is a tool for automatically proving (non-)confluence of logically constrained rewrite systems (LCTRSs). The development of **crest** started in the beginning of 2023 as part of the ARI project<sup>1</sup> and initial experiments were presented at CADE-29 in 2023 [4]. More information and executables of **crest** can be found at

<http://cl-informatik.uibk.ac.at/software/crest/>

Currently, **crest** supports (non-)confluence [2, 4, 7] and limited (non-)termination analysis [2, 1]. The confluence methods mostly rely on closure properties of (parallel) critical pairs. In addition, we support two recent transformation techniques: splitting constrained critical pairs (CCPs) and merging constrained rewrite rules [5]. Furthermore, a constrained variant of the redundant rules technique [3] is implemented [6].

Our tool **crest** participates in the LCTRS category of CoCo 2025.

## References

- [1] Cynthia Kop. Termination of LCTRSs. *CoRR*, abs/1601.03206, 2016. doi: <https://doi.org/10.48550/ARXIV.1601.03206>.
- [2] Cynthia Kop and Naoki Nishida. Term rewriting with logical constraints. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *Proc. 9th International Symposium on Frontiers of Combining Systems*, volume 8152 of *Lecture Notes in Artificial Intelligence*, pages 343–358, 2013. doi: [https://doi.org/10.1007/978-3-642-40885-4\\_24](https://doi.org/10.1007/978-3-642-40885-4_24).
- [3] Julian Nagele, Bertram Felgenhauer, and Aart Middeldorp. Improving automatic confluence analysis of rewrite systems by redundant rules. In Maribel Fernández, editor, *Proc. 26th International Conference on Rewriting Techniques and Applications*, volume 36 of *Leibniz International Proceedings in Informatics*, pages 257–268, 2015.
- [4] Jonas Schöpf and Aart Middeldorp. Confluence criteria for logically constrained rewrite systems. In Brigitte Pientka and Cesare Tinelli, editors, *Proc. 29th International Conference on Automated Deduction*, volume 14132 of *Lecture Notes in Artificial Intelligence*, pages 474–490, 2023. doi: [https://doi.org/10.1007/978-3-031-38499-8\\_27](https://doi.org/10.1007/978-3-031-38499-8_27).
- [5] Jonas Schöpf and Aart Middeldorp. Automated analysis of logically constrained rewrite systems using **crest**. In Arie Gurfinkel and Marijn Heule, editors, *Proc. 31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 15696 of *Lecture Notes in Computer Science*, pages 124–144, 2025.
- [6] Jonas Schöpf and Aart Middeldorp. Improving confluence analysis for LCTRSs. In *Proc. 14th International Workshop on Confluence*, 2025. This volume.
- [7] Jonas Schöpf, Fabian Mitterwallner, and Aart Middeldorp. Confluence of logically constrained rewrite systems revisited. In Christoph Benzmüller, Marijn J.H. Heule, and Renate A. Schmidt, editors, *Proc. 12th International Joint Conference on Automated Reasoning*, volume 14740 of *Lecture Notes in Artificial Intelligence*, pages 298–316, 2024.

---

\*This research is funded by the Austrian Science Fund (FWF) project I 5943-N.

<sup>1</sup><https://ari-informatik.uibk.ac.at/>

# Natto: a small infeasibility prover based on term orders

Teppei Saito

JAIST, Japan

Natto is a prototype tool for infeasibility analysis, primarily developed to assess the impact of formalizing order-based infeasibility methods [1] in the context of certification. It implements a subset of the infeasibility techniques used in the Nagoya Termination Tool [2, 3], specifically polynomial interpretations over the positive and negative integers, as well as the weighted path order. The generated proofs are verifiable by **CeTA**.

## References

- [1] D. Kim, T. Saito, R. Thiemann, and A. Yamada. An Isabelle formalization of co-rewrite pairs for non-reachability in term rewriting. In *Proc. 14th CPP*, pages 272–282, 2025.
- [2] A. Yamada, K. Kusakari, and T. Sakabe. Nagoya Termination Tool. In *Proc. RTA-TLCA 2014*, LNCS 8560, pages 466–475, 2014.
- [3] A. Yamada. Term orderings for non-reachability of (conditional) rewriting. In *Proc. 11th IJCAR*, LNAI 13385, pages 248–267, 2022.

# AProVE25: Confluence Analysis in a Termination Tool

Jan-Christoph Kassing<sup>1</sup> and Tobias Sokolowski<sup>2</sup>

<sup>1</sup> RWTH Aachen University, Aachen, Germany  
Kassing@cs.rwth-aachen.de

<sup>2</sup> RWTH Aachen University, Aachen, Germany  
Tobias.Sokolowski@rwth-aachen.de

AProVE (Automated Program Verification Environment) is a tool for fully automatic program verification. Its primary focus is on proving termination, analyzing (worst-case) complexity, and verifying safety or infeasibility of different programming languages including term rewriting. For further details on AProVE's general approach for these analyses, see [3].

Termination and confluence are two closely related properties. Local confluence allows us to reduce the analysis of termination from arbitrary rewrite sequences to innermost rewrite sequences, a task that has been shown to be substantially easier. Moreover, the dependency graph heavily relies on proving infeasibility so that better computable approximations may yield a more exact model of the actual dependency graph. On the other hand, confluence is a decidable property if termination is guaranteed. Therefore, AProVE has already implemented some techniques for confluence, and we want to present the power of the currently implemented methods and, in the future, to improve confluence and reachability analysis within AProVE.

AProVE relies on three main techniques:

- *Termination-based Confluence Analysis:* We use our termination analysis to check whether the well-known decision procedure for confluence is applicable.
- *Modularity:* We use basic results on modularity of confluence from the last century. To be precise, we implemented different modularity results mentioned in [1] and [4].
- *Joinability and Reachability Analysis:* To disprove joinability of critical pairs, AProVE uses techniques originally designed for proving infeasibility within dependency graph approximations, e.g., checking for unifiability between the target term and an approximation of the top part of the source term that remains the same during rewrite steps.

In the future, we want to investigate the problem of confluence within the probabilistic setting, a question raised in recent years and investigated in, e.g., [2], but which has received relatively little attention. Due to the complex interplay of probabilities and non-deterministic choices, we believe that this is an interesting direction for the confluence community.

## References

- [1] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, Cambridge, 1998.
- [2] Alejandro Díaz-Caro and Guido Martínez. Confluence in Probabilistic Rewriting. *Electronic Notes in Theoretical Computer Science*, 338:115–131, October 2018.
- [3] 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.
- [4] Enno Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, New York, NY, 2002.

# Hakusan 0.12: A Confluence Tool

Fuyuki Kawano, Hiroka Hondo, Nao Hirokawa, and Kiraku Shintani

JAIST, Japan

{f-kawano,s2410171,hirokawa}@jaist.ac.jp, s.kiraku@gmail.com

Hakusan (<https://www.jaist.ac.jp/project/saigawa/>) is a confluence tool for left-linear term rewrite systems (TRSs). It analyzes confluence by successive application of *rule removal* criteria [5, 6, 10] based on rule labeling [9, 13], critical pair systems [4], and the generalization of Knuth and Bendix' criterion by Klein and Hirokawa [7]. Hakusan can produce proof certificates verifiable by CeTA [11], see [3].

Compared to the last version of Hakusan [6], non-confluence analysis has been improved by adopting the approach used in CSI [8]. Given a TRS  $\mathcal{R}$  and convertible terms  $t_1 \leftrightarrow_{\mathcal{R}}^* t_2$ , this approach constructs two tree automata  $\mathcal{A}_1$  and  $\mathcal{A}_2$  such that  $L(\mathcal{A}_i)$  is closed under  $\mathcal{R}$ -rewriting and  $t_i \in L(\mathcal{A}_i)$  for each  $i$ . If  $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) = \emptyset$  then  $t_1$  and  $t_2$  are not joinable. Thus, non-confluence of  $\mathcal{R}$  is concluded. Construction of  $\mathcal{A}_i$  is heuristically done by tree automata completion [2] and closedness of its language under rewriting is tested by checking state compatibility and state coherence [1]. Tree automata completion uses the following criterion for testing closedness under rewriting.

**Proposition 1.** *Let  $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$  be a deterministic tree automaton and  $\mathcal{R}$  a TRS. The language  $L(\mathcal{A})$  is closed under  $\mathcal{R}$ -rewriting if  $\mathcal{A}$  is compatible with all rules in  $\mathcal{R}$ . Here compatibility with  $\ell \rightarrow r$  means that  $r\sigma \rightarrow_{\Delta}^* q$  whenever  $\sigma : \text{Var}(\ell) \rightarrow Q$  and  $\ell\sigma \rightarrow_{\Delta}^* q \in Q$ .*

**Example 1.** *Consider the TRS  $\mathcal{R}$  over the signature  $\mathcal{F} = \{h, f, g, a, b\}$ :*

$$1: h(g, a, a) \rightarrow h(f, a, a) \quad 2: h(x, b, y) \rightarrow h(x, y, y) \quad 3: f \rightarrow g \quad 4: a \rightarrow b$$

By the aforementioned approach we can prove that the convertible terms  $t_1 = h(g, b, b)$  and  $t_2 = h(g, a, a)$  are not joinable. Here we illustrate how compatibility for Proposition 1 is tested. Consider the deterministic tree automaton  $\mathcal{A}_1 = (\mathcal{F}, Q, \{2\}, \Delta)$  with  $Q = \{0, 1, 2\}$  and  $\Delta = \{g \rightarrow 0, b \rightarrow 1, h(0, 1, 1) \rightarrow 2\}$ . This automaton satisfies  $L(\mathcal{A}_1) = \{t_1\}$  and the compatibility with all rules in  $\mathcal{R}$ . For example, the compatibility with rule 2 is verified by checking that all triples  $(q_1, q_2, q_3) \in Q^3$  satisfy the implication  $h(q_1, b, q_2) \rightarrow_{\Delta}^* q_3 \implies h(q_1, q_2, q_2) \rightarrow_{\Delta}^* q_3$ .

Compatibility check is a major bottleneck in tree automata completion. To ease the check, our tool exploits *persistence* of reachability.

**Proposition 2.** *Let  $\mathcal{R}$  be a many-sorted TRS and  $\Theta$  a sort elimination-operator [12]. If  $s$  is well-sorted then  $s \rightarrow_{\mathcal{R}}^* t$  and  $\Theta(s) \rightarrow_{\Theta(\mathcal{R})}^* \Theta(t)$  are equivalent.*

Thus, we can analyze reachability after performing type introduction. Type discipline enables us to discard ill-sorted state substitutions for compatibility check.

**Example 2** (continued from Example 1). *The TRS  $\mathcal{R}$  and the terms  $t_1$  and  $t_2$  can be seen as a TRS and terms over the many-sorted signature:*

$$h : A \times B \times B \rightarrow C \quad f : A \quad g : A \quad a : B \quad b : B$$

*The sorted signature naturally assigns sorts to the states of  $\mathcal{A}_1$  as follows:*

$$0 : A \quad 1 : B \quad 2 : C$$

*As  $h(q_1, b, q_2) \rightarrow_{\Delta}^* q_3$  induces  $q_1 : A$ ,  $q_2 : B$ , and  $q_3 : C$ , the compatibility of  $\mathcal{A}$  with rule 2 follows by testing the previous implication only for  $(q_1, q_2, q_3) \in \{0\} \times \{1\} \times \{2\}$ .*

## References

- [1] B. Felgenhauer and R. Thiemann. Reachability, confluence, and termination analysis with state-compatible automata. *Information and Computation*, 253:467–483, 2017.
- [2] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proceedings of 9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *LNCS*, pages 151–165, 1998. doi:[doi.org/10.1007/BFb0052368](https://doi.org/10.1007/BFb0052368).
- [3] N. Hirokawa, D. Kim, K. Shintani, and R. Thiemann. Certification of confluence- and commutation-proofs via parallel critical pairs. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 147–161, 2024. doi:[10.1145/3636501.3636949](https://doi.org/10.1145/3636501.3636949).
- [4] N. Hirokawa and A. Middeldorp. Decreasing diagrams and relative termination. *Journal of Automated Reasoning*, 47:481–501, 2011. doi:[10.1007/s10817-011-9238-x](https://doi.org/10.1007/s10817-011-9238-x).
- [5] N. Hirokawa and K. Shintani. Rule removal for confluence. In *Proceedings of 13th International Workshop on Confluence*, pages 50–54, 2024.
- [6] F. Kawano, N. Hirokawa, and K. Shintani. Hakusan 0.11: A confluence tool. In *Proceedings of 13th International Workshop on Confluence*, pages 67–68, 2024.
- [7] D. Klein and N. Hirokawa. Confluence of non-left-linear TRSs via relative termination. In *Proceedings of 18th International Conference on Logic Programming and Automated Reasoning*, volume 7180 of *LNCS*, pages 258–273, 2012. doi:[10.1007/978-3-642-28717-6\\_21](https://doi.org/10.1007/978-3-642-28717-6_21).
- [8] J. Nagele, B. Felgenhauer, and A. Middeldorp. Csi: New evidence — a progress report. In *Proceedings of 26th International Conference on Automated Deduction*, pages 385–397, 2017. doi:[10.1007/978-3-319-63046-5\\_24](https://doi.org/10.1007/978-3-319-63046-5_24).
- [9] V. van Oostrom. Confluence by decreasing diagrams, converted. In *Proceedings of 19th International Conference on Rewriting Techniques and Applications*, volume 5117 of *LNCS*, pages 306–320, 2008. doi:[10.1007/978-3-540-70590-1\\_21](https://doi.org/10.1007/978-3-540-70590-1_21).
- [10] K. Shintani and N. Hirokawa. Compositional confluence criteria. *Logical Methods in Computer Science*, 20:6:1–6:28, 2024. doi:[10.46298/lmcs-20\(1:6\)2024](https://doi.org/10.46298/lmcs-20(1:6)2024).
- [11] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *LNCS*, pages 452–468, 2009.
- [12] J. van de Pol. Modularity in many-sorted term rewriting systems. Technical report, Utrecht University, 1992. Master’s thesis.
- [13] H. Zankl, B. Felgenhauer, and A. Middeldorp. Labelings for decreasing diagrams. *Journal of Automated Reasoning*, 54(2):101–133, 2015. doi:[10.1007/s10817-014-9316-y](https://doi.org/10.1007/s10817-014-9316-y).



# CONFident at the 2025 Confluence Competition\*

Raúl Gutiérrez and Salvador Lucas

VRAIN, Universitat Politècnica de València, Valencia, Spain  
raguti@upv.es  
slucas@dsic.upv.es

## 1 Overview

CONFident is a tool which is able to prove confluence of TRSs, CS-TRSs, CTRSs and CS-CTRSs. The tool is available here:

<http://zenon.dsic.upv.es/confident/>.

It is written in Haskell implementing the Confluence Framework. We implement the processors using the logical approach presented in [1, 3, 6] and mechanizing them by external tools like MU-TERM [3], infChecker [1, 5], AGES [2], Prover9 and Mace4 [8] and Barcellogic<sup>1</sup>.

In 2025, CONFident was enhanced with the new techniques implemented in infChecker, improved with some relations needed in [7] and now supports the ARI format natively.

## References

- [1] R. Gutiérrez and S. Lucas. Automatically Proving and Disproving Feasibility Conditions. In N. Peltier and V. Sofronie-Stokkermans, editor, *Proc. of IJCAR'2020*, LNCS 12167:416–435. Springer, 2020.
- [2] R. Gutiérrez and S. Lucas. Automatic Generation of Logical Models with AGES. In *CADE 2019: Automated Deduction - CADE 27*, LNCS 11716:287:299. Springer, 2019.
- [3] R. Gutiérrez and S. Lucas. MU-TERM: Verify Termination Properties Automatically (System Description). In N. Peltier and V. Sofronie-Stokkermans, editor, *Proc. of IJCAR'2020*, LNCS 12167:436–447. Springer, 2020.
- [4] R. Gutiérrez. and S. Lucas. Proving Confluence in the Confluence Framework with CONFident. *Fundamenta Informaticae*, 192, Issue 2: LOPSTR 2022, 2024.
- [5] R. Gutiérrez and S. Lucas. Proving and disproving feasibility with infChecker. In *Proc. of the 14th International Workshop on Confluence, IWC'25*, to appear, 2025.
- [6] S. Lucas. Proving semantic properties as first-order satisfiability. *Artificial Intelligence* 277, paper 103174, 24 pages, 2019.
- [7] S. Lucas. Confluence of Almost Parallel-Closed Generalized Term Rewriting Systems. In *Proc. of 30th International Conference on Automated Deduction, CADE-30*, to appear, 2025.
- [8] W. McCune. Prover9 and Mace4. [online]. Available at <https://www.cs.unm.edu/~mccune/mace4/>.

---

\*Supported by MCIN/AEI/10.13039/501100011033 and by “ERDF A way of making Europe” (FEDER), (PID2021-122830OB-C42 and PID2021-122830OB-C44) and by the grant CIPROM/2022/6 funded by Generalitat Valenciana

<sup>1</sup><https://barcellogic.com/>

# infChecker at the 2025 Confluence Competition\*

Raúl Gutiérrez and Salvador Lucas

VRAIN, Universitat Politècnica de València, Valencia, Spain  
raguti@upv.es  
slucas@dsic.upv.es

## 1 Overview

infChecker is a tool for checking *(in)feasibility* of sequences of rewrite and relations with respect to *first-order theories*, called goals [5]. infChecker participates in the INF category at the Confluence Competition but it is also used as an external tool in CONFident, which participates in several categories in the Competition.

The tool is available here:

<http://zenon.dsic.upv.es/infChecker/>.

Some processors are mechanized using external tools like AGES [2], Prover9 and Mace4 [6]. Latest description of the tool can be found in [1, 3].

In 2025, we extended the tool (and its input format) to support *Generalized Term Rewriting Systems* (GTRSs) [4], which are CTRS extended with a replacement map  $\mu$  and by allowing more general conditions in rules, atoms defined by a set of Horn clauses [3]. Furthermore, we now support problems in ARI format, which in previous years were handled by external tools.

## References

- [1] R. Gutiérrez and S. Lucas. Automatically Proving and Disproving Feasibility Conditions. In N. Peltier and V. Sofronie-Stokkermans, editor, *Proc. of IJCAR'2020*, LNCS 12167:416–435. Springer, 2020.
- [2] R. Gutiérrez and S. Lucas. Automatic Generation of Logical Models with AGES. In *CADE 2019: Automated Deduction - CADE 27*, LNCS 11716:287:299. Springer, 2019.
- [3] R. Gutiérrez and S. Lucas. Proving and disproving feasibility with infChecker. In *Proc. of the 14th International Workshop on Confluence, IWC'25*, to appear, 2025.
- [4] S. Lucas. Local confluence of conditional and generalized term rewriting systems. *Journal of Logical and Algebraic Methods in Programming*, 136, paper 100926, pages 1-23, 2024.
- [5] S. Lucas and R. Gutiérrez. Use of Logical Models for Proving Infeasibility in Term Rewriting. *Information Processing Letters*, 136:90–95, 2018.
- [6] W. McCune. Prover9 and Mace4. [online]. Available at <https://www.cs.unm.edu/~mccune/mace4/>.

---

\*Supported by MCIN/AEI/10.13039/501100011033 and by “ERDF A way of making Europe” (PID2021-122830OB-C42 and PID2021-122830OB-C44) and by the grant CIPROM/2022/6 funded by Generalitat Valenciana

# CoCo 2025 Participant: CeTA 3.6

Christina Kirk and René Thiemann

University of Innsbruck, Austria

The tool CeTA [6] is a certifier for, among other properties, (non-)confluence of term rewrite systems with and without conditions. Its soundness is proven as part of the formal proof library *IsaFoR*, the Isabelle Formalization of Rewriting. Below, we present the relevant changes from last year’s version (3.1) to this year’s version (3.6). For a complete reference of supported techniques we refer to the certification problem format (CPF) and the *IsaFoR/CeTA* website:

<http://cl-informatik.uibk.ac.at/isafor/>

CeTA 3.6 has the new feature that it is capable of checking *feasibility* proofs, so that now in the INF category of CoCo both YES-answers and NO-answers can be certified. Essentially, a certificate of a feasibility proof consists of the substitution that proves feasibility in combination with details on the rewrite sequence, consisting of conditional rewrite steps.

In CeTA 3.6, also a new class of term orderings have been added. These orderings can be used in non-joinability proofs as discrimination pairs, or in infeasibility proofs as co-rewrite pairs. The new class of term orderings are Hofbauer and Waldmann’s core matrix interpretations [2]. We generalized these orderings from the SRS version to a full TRS version [4]. Note that core matrix interpretations have slightly different requirements than the matrix interpretations of Endrullis et al. [1].

Regarding non-commutation of two TRSs  $\mathcal{R}$  and  $\mathcal{S}$ , CeTA 3.6 has added a swap technique, so that the role of  $\mathcal{R}$  and  $\mathcal{S}$  can be swapped. The reason is that some non-commutation techniques are not symmetric. Previously, swapping was only supported within commutation proofs.

A further significant addition has been added to *IsaFoR*, namely in the form of Okui’s confluence criterion [3, 5]. However, this part is not yet available in CeTA: it remains to develop and verify an algorithm to compute all simultaneous critical pairs of TRSs.

## References

- [1] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. *J. Autom. Reason.*, 40(2-3):195–220, 2008. URL: <https://doi.org/10.1007/s10817-007-9087-9>, doi:10.1007/S10817-007-9087-9.
- [2] Dieter Hofbauer and Johannes Waldmann. Termination of string rewriting with matrix interpretations. In Frank Pfenning, editor, *Term Rewriting and Applications, 17th International Conference, RTA 2006, Seattle, WA, USA, August 12-14, 2006, Proceedings*, volume 4098 of *Lecture Notes in Computer Science*, pages 328–342. Springer, 2006. doi:10.1007/11805618\_25.
- [3] Christina Kirk and Aart Middeldorp. Formalizing simultaneous critical pairs for confluence of left-linear rewrite systems. In Kathrin Stark, Amin Timany, Sandrine Blazy, and Nicolas Tabareau, editors, *Proceedings of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2025, Denver, CO, USA, January 20-21, 2025*, pages 156–170. ACM, 2025. doi:10.1145/3703595.3705881.
- [4] Ulysse Le Huitouze and René Thiemann. Core matrix interpretations for proving termination of term rewrite systems. In Dieter Hofbauer and Johannes Waldmann, editors, *Proceedings of the 20th International Workshop on Termination (WST 2025)*, 2025. To appear.

- [5] Satoshi Okui. Simultaneous critical pairs and church-rosser property. In Tobias Nipkow, editor, *Rewriting Techniques and Applications, 9th International Conference, RTA-98, Tsukuba, Japan, March 30 - April 1, 1998, Proceedings*, volume 1379 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 1998. URL: <https://doi.org/10.1007/BFb0052357>, doi:10.1007/BFb0052357.
- [6] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *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*, pages 452–468. Springer, 2009. doi:10.1007/978-3-642-03359-9\_31.

# ACP: System Description for CoCo 2025

Takahito Aoto

Faculty of Engineering, Niigata University  
aoto@ie.niigata-u.ac.jp

ACP (Automated Confluence Prover) is a tool for proving confluence and some related properties of (conditional) term rewriting systems. Below we provide a brief overview of ACP.

A primary functionality of ACP is proving confluence (CR) of term rewriting systems (TRSs). ACP integrates multiple direct criteria for guaranteeing confluence of TRSs. It also incorporates divide-and-conquer criteria by which confluence or non-confluence of TRSs can be inferred from those of their components. Several methods for disproving confluence are also employed. For some criteria, it supports generation of proofs in CPF format that can be certified by certifiers. The internal structure of the prover is kept simple and is mostly inherited from the version 0.11a, which has been described in [3]. It also deal with confluence of oriented conditional term rewriting systems. Besides confluence, ACP supports proving the UNC property (unique normal form property w.r.t. conversion), the UNR property (unique normal form property w.r.t. reduction), and the commutation property of term rewriting systems [2, 4, 5, 6]. Some commutativity (dis)proving proofs have capability of producing certifiable proofs in CPF format.

ACP is written in Standard ML of New Jersey (SML/NJ) and the source code is also available from [1]. It uses a SAT prover such as MiniSAT and an SMT prover YICES as external provers. It internally contains an automated (relative) termination prover for TRSs but external (relative) termination provers can be substituted optionally. Users can specify criteria to be used so that each criterion or any combination of them can be tested. Several levels of verbosity are available for the output so that users can investigate details of the employed approximations for each criterion or can get only the final result of the prover's attempt.

## References

- [1] ACP (Automated Confluence Prover). <http://www.nue.ie.niigata-u.ac.jp/tools/acp/>.
- [2] T. Aoto and Y. Toyama. Automated proofs of unique normal forms w.r.t. conversion for term rewriting systems. In *Proc. of 12th FroCoS*, volume 11715 of *LNAI*, pages 330–347. Springer-Verlag, 2019.
- [3] T. Aoto, J. Yoshida, and Y. Toyama. Proving confluence of term rewriting system automatically. In *Proc. of 20th RTA*, volume 5595 of *LNCS*, pages 93–102. Springer-Verlag, 2009.
- [4] T. Aoto. Proving uniqueness of normal forms w.r.t. reduction of term rewriting systems. In *Proc. of 34th LOPSTR*, volume 14919 of *LNCS*, pages 185–201. Springer-Verlag, 2024.
- [5] M. Yamaguchi and T. Aoto. A fast decision procedure for uniqueness of normal forms w.r.t. conversion of shallow term rewriting systems. In *Proc. of 5th FSCD*, volume 167 of *LIPIcs*, pages 9:1–9:23. Schloss Dagstuhl, 2020.
- [6] J. Yoshida, T. Aoto, and Y. Toyama. Automating confluence check of term rewriting systems. *Computer Software*, 26(2):76–92, 2009.

# CO3 (Version 2.6)

Naoki Nishida and Misaki Kojima

Nagoya University, Nagoya, Japan  
nishida@i.nagoya-u.ac.jp   k-misaki@nagoya-u.jp

CO3, a **converter** for proving **confluence** of **conditional** TRSs,<sup>1</sup> tries to prove confluence of conditional term rewrite systems (CTRSs, for short) by using a transformational approach (cf. [8]). The tool first transforms a given weakly-left-linear (WLL, for short) 3-DCTRS into an unconditional term rewrite system (TRS, for short) by using  $\mathbb{U}_{conf}$  [3], a variant of the *unraveling*  $\mathbb{U}$  [10], and then verifies confluence of the transformed TRS by using the following theorem: A 3-DCTRS  $\mathcal{R}$  is confluent if  $\mathcal{R}$  is WLL and  $\mathbb{U}_{conf}(\mathcal{R})$  is confluent [2, 3]. The tool is very efficient because of very simple and lightweight functions to verify properties such as confluence and termination of TRSs.

Since version 2.0, a *narrowing-tree*-based approach [9, 4] to prove infeasibility of a condition w.r.t. a CTRS has been implemented [5]. The approach is applicable to *syntactically deterministic* CTRSs that are operationally terminating and *ultra-right-linear* w.r.t. the *optimized* unraveling. To prove infeasibility of a condition  $c$ , the tool first proves confluence, and then linearizes  $c$  if failed to prove confluence; then, the tool computes and simplifies a narrowing tree for  $c$ , and examines the emptiness of the narrowing tree. Since version 2.2, CO3 accepts both *join* and *semi-equational* CTRSs, and transforms them into equivalent DCTRSs to prove confluence or infeasibility [6].

The difference from the previous version [7] is a slight improvement of the subterm criterion. CO3 uses very lightweight criteria for proving termination, while using the DP framework [1]. The *subterm criterion* implemented in the previous version considers the first argument of marked symbols, while arbitrary arguments can be taken. This version uses the second argument in addition to the first one: The subterm criterion processor tries to prove finiteness of a given DP problem by means of the first argument, and if failed, then it tries it by means of the second argument. This slight improvements succeeds in proving termination of (C)TRSs and thus confluence of, e.g., 1009.ari.

## References

- [1] J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In F. Baader and A. Voronkov, editors, *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3452 of *Lecture Notes in Computer Science*, pages 301–331. Springer, 2005.
- [2] K. Gmeiner, B. Gramlich, and F. Schernhammer. On soundness conditions for unraveling deterministic conditional rewrite systems. In A. Tiwari, editor, *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications*, volume 15 of *Leibniz International Proceedings in Informatics*, pages 193–208. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.
- [3] K. Gmeiner, N. Nishida, and B. Gramlich. Proving confluence of conditional term rewriting systems via unravelings. In N. Hirokawa and V. van Oostrom, editors, *Proceedings of the 2nd International Workshop on Confluence*, pages 35–39, 2013.
- [4] Y. Maeda, N. Nishida, M. Sakai, and T. Kobayashi. Extending narrowing trees to basic narrowing in term rewriting. IEICE Technical Report SS2018-39, the Institute of Electronics, Information and Communication Engineers, 2019. Vol. 118, No. 385, pp. 73–78, in Japanese.

---

<sup>1</sup><http://www.trs.css.i.nagoya-u.ac.jp/co3/>

- [5] N. Nishida. CO3 (Version 2.1). In M. Ayala-Rincón and S. Mimram, editors, *Proceedings of the 9th International Workshop on Confluence*, page 67, 2020.
- [6] N. Nishida. CO3 (Version 2.2). In S. Mimram and C. Rocha, editors, *Proceedings of the 10th International Workshop on Confluence*, page 61, 2021.
- [7] N. Nishida and M. Kojima. CO3 (Version 2.5). In C. Chenavier and N. Nishida, editors, *Proceedings of the 13th International Workshop on Confluence*, pages 71–72, 2024.
- [8] N. Nishida, T. Kuroda, and K. Gmeiner. CO3 (Version 1.3). In B. Accattoli and A. Tiwari, editors, *Proceedings of the 5th International Workshop on Confluence*, page 74, 2016.
- [9] N. Nishida and Y. Maeda. Narrowing trees for syntactically deterministic conditional term rewriting systems. In H. Kirchner, editor, *Proceedings of the 3rd International Conference on Formal Structures for Computation and Deduction*, volume 108 of *Leibniz International Proceedings in Informatics*, pages 26:1–26:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- [10] E. Ohlebusch. Termination of logic programs: Transformational methods revisited. *Applicable Algebra in Engineering, Communication and Computing*, 12(1/2):73–116, 2001.

# CRaris (Version 1.1)

Naoki Nishida and Misaki Kojima

Nagoya University, Nagoya, Japan  
nishida@i.nagoya-u.ac.jp    kojima@i.nagoya-u.ac.jp

CRaris, a **CR** checker for LCTRSs in **ARI** style,<sup>1</sup> is a tool to prove confluence of *logically constrained term rewrite systems* (LCTRSs, for short) [5] written in ARI format [1].<sup>2</sup> The tool is based on Crisys2, **constrained rewriting induction system** (version 2),<sup>3</sup> and receives LCTRSs written in ARI format only to prove confluence, while Crisys2 has many functions to e.g., solve *all-path reachability problems* [3]. To prove confluence of LCTRSs, the tool uses the following criteria:

- weak orthogonality [5], and
- termination and joinability of critical pairs [8].

To prove termination, the tool uses the DP framework for LCTRSs [4] without any interpretation method, together with a criterion for LCTRSs with bitvector arithmetics [6].

The *critical pairs* of two constrained rewrite rules  $\rho_1 : \ell_1 \rightarrow r_1 [\varphi_1]$  and  $\rho_2 : \ell_2 \rightarrow r_2 [\varphi_2]$  with distinct variables (i.e.,  $\text{Var}(\ell_1, r_1, \varphi_1) \cap \text{Var}(\ell_2, r_2, \varphi_2) = \emptyset$ ) are all tuples  $\langle s, t, \phi \rangle$  such that a non-variable subterm  $\ell_1|_p$  of  $\ell_1$  at a position  $p$  is unifiable with  $\ell_2$ , “ $p \neq \varepsilon$ ,  $\rho_1 \neq \rho_2$  up to variable renaming, or  $\text{Var}(r_1) \subseteq \text{Var}(\ell_1)$ ”, the most general unifier  $\gamma$  of  $\ell_1|_p$  and  $\ell_2$  respects variables of both  $\rho_1$  and  $\rho_2$ , i.e.,  $\gamma(x)$  is either a value or a variable for all variables  $x$  in  $\text{Var}(\varphi_1, \varphi_2) \cup (\text{Var}(r_1, r_2) \setminus \text{Var}(\ell_1, \ell_2))$ ,  $(\varphi_1 \wedge \varphi_2)\gamma$  is satisfiable,  $s = r_1\gamma$ ,  $t = (\ell_1[r_2]_p)\gamma$ , and  $\phi = (\varphi_1 \wedge \varphi_2)\gamma$ . The set of critical pairs of an LCTRS  $\mathcal{R}$  is denoted by  $CP(\mathcal{R})$ , which includes all critical pairs of two rules in  $\mathcal{R} \cup \mathcal{R}_{calc}$ . A critical pair  $\langle s, t, \phi \rangle$  is called *trivial* if  $s[\phi] \sim t[\phi]$ . An LCTRS  $\mathcal{R}$  is called *weakly orthogonal* if  $\mathcal{R}$  is left-linear and all critical pairs of  $\mathcal{R}$  are trivial.

**Theorem 1** ([5]). *A weakly orthogonal LCTRS is confluent.*

A critical pair  $\langle s, t, \phi \rangle$  is called *joinable* if  $(\langle s, t \rangle [\phi]) \rightarrow_{\mathcal{R}}^* (\langle s', t' \rangle [\phi'])$  and  $s'[\phi'] \sim t'[\phi']$ .

**Theorem 2** ([8]). *A terminating LCTRS is confluent if all its critical pairs are joinable.*

The previous version [7] uses syntactic equivalence of terms as a sufficient condition for a critical pair  $\langle s, t, \phi \rangle$  being trivial.

**Proposition 3** ([7]). *A critical pair  $\langle s, s, \phi \rangle$  is trivial and thus joinable.*

This version uses the idea of EQ-DELETION of constrained rewriting induction [2].

**Proposition 4.** *A critical pair  $\langle s, t, \phi \rangle$  is trivial if there exist positions  $p_1, \dots, p_n$  of  $s$  such that  $p_1, \dots, p_n$  are positions of  $t$ ,  $t = s[t_1, \dots, t_n]_{p_1, \dots, p_n}$ ,  $s|_{p_1}, t_1, \dots, s|_{p_n}, t_n$  are theory terms,  $\text{Var}(s|_{p_1}, \dots, s|_{p_n}, t_1, \dots, t_n) \subseteq \text{Var}(\phi)$ , and  $\phi \wedge \neg(\bigwedge_{i=1}^n (s|_{p_i} = t_i))$  is unsatisfiable.*

In addition, this version uses a very simple variant of the disproof criterion—an LCTRS is not confluent if there exists a constrained critical pair that rewrites to a non-trivial constrained equation in normal form—in [9, Lemma 1].

**Proposition 5.** *An LCTRS  $\mathcal{R}$  is not confluent if there exists a critical pair  $\langle s, t, \phi \rangle$  of  $\mathcal{R}$  such that  $s, t$  are variables in  $\text{Var}(\phi)$  and  $\phi \wedge \neg(\bigwedge_{i=1}^n (s|_{p_i} = t_i))$  is satisfiable.*

<sup>1</sup><http://www.trs.css.i.nagoya-u.ac.jp/craris/>

<sup>2</sup><https://project-coco.uibk.ac.at/ARI/lctrs.php>

<sup>3</sup><https://www.trs.cm.is.nagoya-u.ac.jp/crisys/>



## References

- [1] T. Aoto, N. Hirokawa, D. Kim, M. Kojima, A. Middeldorp, F. Mitterwallner, N. Nishida, T. Saito, J. Schöpfung, K. Shintani, R. Thiemann, and A. Yamada. A new format for rewrite systems. In C. Chenavier and S. Winkler, editors, *Proceedings of the 12th International Workshop on Confluence*, pages 32–37, 2023.
- [2] C. Fuhs, C. Kop, and N. Nishida. Verifying procedural programs via constrained rewriting induction. *ACM Transactions on Computational Logic*, 18(2):14:1–14:50, 2017.
- [3] M. Kojima and N. Nishida. Reducing non-occurrence of specified runtime errors to all-path reachability problems of constrained rewriting. *Journal of Logical and Algebraic Methods in Programming*, 135:1–19, 2023.
- [4] C. Kop. Termination of LCTRSs. In *Proceedings of the 13th International Workshop on Termination*, pages 1–5, 2013.
- [5] C. Kop and N. Nishida. Term rewriting with logical constraints. In P. Fontaine, C. Ringeissen, and R. A. Schmidt, editors, *Proceedings of the 9th International Symposium on Frontiers of Combining Systems*, volume 8152 of *Lecture Notes in Artificial Intelligence*, pages 343–358. Springer, 2013.
- [6] A. Matsumi, N. Nishida, M. Kojima, and D. Shin. On singleton self-loop removal for termination of LCTRSs with bit-vector arithmetic. In A. Yamada, editor, *Proceedings of the 19th International Workshop on Termination*, pages 1–6, 2023.
- [7] N. Nishida and M. Kojima. CRaris: CR checker for LCTRSs in ARI Style. In C. Chenavier and N. Nishida, editors, *Proceedings of the 13th International Workshop on Confluence*, pages 83–84, 2024.
- [8] J. Schöpfung and A. Middeldorp. Confluence criteria for logically constrained rewrite systems. In B. Pientka and C. Tinelli, editors, *Proceedings of the 29th International Conference on Automated Deduction*, volume 14132 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2023.
- [9] J. Schöpfung and A. Middeldorp. Automated analysis of logically constrained rewrite systems using crest. In A. Gurfinkel and M. Heule, editors, *Proceedings of the 31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 15696 of *Lecture Notes in Computer Science*, pages 124–144. Springer, 2025.

# CoCo 2025 Participant: FORT-h 2.1

Fabian Mitterwallner and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria  
fabian.mitterwallner@student.uibk.ac.at, aart.middeldorp@uibk.ac.at

The first-order theory of rewriting is a decidable theory for finite left-linear right-ground rewrite systems. The decision procedure goes back to Dauchet and Tison [2]. FORT-h is a reimplement of the tool FORT [5], but is based on a new variant of the decision procedure, described in [3], for the larger class of linear variable-separated rewrite systems. This variant supports a more expressive theory and is based on anchored ground tree transducers. More importantly, it can produce certificates for the YES/NO answers. These certificates can then be verified by FORTify, an independent Haskell program that is code-generated from the formalization of the decision procedure in the proof assistant Isabelle/HOL.

A command-line version of FORT-h can be downloaded from

[http://fortissimo.uibk.ac.at/fort\(ify\)/](http://fortissimo.uibk.ac.at/fort(ify)/)

FORT-h participates in the TRS category of CoCo 2025 both as a standalone tool and in combination with FORTify [4] to produce certified YES/NO answers. In 2024 FORT-h tied ACP [1] for the most YES answers in the COM category. Moreover, it won the RELIABILITY<sup>1</sup> category, by producing the most certifiable answers across all categories.

FORT-h 2.1 accepts input problems in ARI<sup>2</sup> format.

## References

- [1] Takahito Aoto. ACP: System Description for CoCo 2024. In *Proc. 13th International Workshop on Confluence*, page 77, 2024.
- [2] Max Dauchet and Sophie Tison. The Theory of Ground Rewrite Systems is Decidable. In *Proc. 5th IEEE Symposium on Logic in Computer Science*, pages 242–248, 1990. doi: [10.1109/LICS.1990.113750](https://doi.org/10.1109/LICS.1990.113750).
- [3] Aart Middeldorp, Alexander Lochmann, and Fabian Mitterwallner. First-Order Theory of Rewriting for Linear Variable-Separated Rewrite Systems: Automation, Formalization, Certification. *Journal of Automated Reasoning*, 67(14):1–76, 2023. doi: [10.1007/s10817-023-09661-7](https://doi.org/10.1007/s10817-023-09661-7).
- [4] Fabian Mitterwallner and Aart Middeldorp. CoCo 2025 Participant: FORTify 2.0. In *Proc. 14th International Workshop on Confluence*, 2025. This volume.
- [5] Franziska Rapp and Aart Middeldorp. FORT 2.0. In *Proc. 9th International Joint Conference on Automated Reasoning*, volume 10900 of *LNCS (LNAI)*, pages 81–88, 2018. doi: [10.1007/978-3-319-94205-6\\_6](https://doi.org/10.1007/978-3-319-94205-6_6).

---

<sup>1</sup><https://project-coco.uibk.ac.at/2024/results.php#certification>

<sup>2</sup><https://project-coco.uibk.ac.at/ARI/>

# CoCo 2025 Participant: CSI 1.2.7

Fabian Mitterwallner and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria  
`fabian.mitterwallner@student.uibk.ac.at`, `aart.middeldorp@uibk.ac.at`

CSI is an automatic tool for (dis)proving confluence and related properties of first-order term rewrite systems (TRSs). It has been in development since 2010 and has had many different contributors over the years. Its name is derived from the Confluence of the rivers Sill and Inn in Innsbruck. The tool is available from

<http://cl-informatik.uibk.ac.at/software/csi>

under a LGPLv3 license. A detailed description of CSI can be found in [4]. Some of the implemented techniques are described in [1,3,5]. CSI can also produce certificates for confluence results, which are checked by CēTA [2].

CSI participates in the TRS category of CoCo 2025 both as a standalone tool and in combination with CēTA providing certified confluence and non-confluence answers. In 2024 CSI won not only the TRS category, but also the NFP, SRS, UNC and UNR categories. Together with CēTA it came in second in the RELIABILITY category where tools are ranked based on the number of certifiable answers.

CSI uses the conversion tool<sup>1</sup> to transform ARI problems into COPS problems.

## References

- [1] Bertram Felgenhauer. Confluence for Term Rewriting: Theory and Automation. PhD thesis, University of Innsbruck, 2015.
- [2] Christina Kirk and René Thiemann. CoCo 2025 Participant: CēTA 3.6. In *Proc. 14th International Workshop on Confluence*, 2025. This volume.
- [3] Julian Nagele. Mechanizing Confluence: Automated and Certified Analysis of First- and Higher-Order Rewrite Systems. PhD thesis, University of Innsbruck, 2017.
- [4] Julian Nagele, Bertram Felgenhauer, and Aart Middeldorp. CSI: New Evidence – A Progress Report. In *Proc. 26th International Conference on Automated Deduction*, volume 10395 of *Lecture Notes in Artificial Intelligence*, pages 385–397, 2017. doi: [10.1007/978-3-319-63046-5\\_24](https://doi.org/10.1007/978-3-319-63046-5_24).
- [5] Harald Zankl. Challenges in Automation of Rewriting. Habilitation thesis, University of Innsbruck, 2014.

---

<sup>1</sup><https://project-coco.uibk.ac.at/ARI/#conversion>

# CoCo 2025 Participant: FORTify 2.1

Fabian Mitterwallner and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria  
fabian.mitterwallner@student.uibk.ac.at, aart.middeldorp@uibk.ac.at

The first-order theory of rewriting is a decidable theory for linear variable-separated rewrite systems. The decision procedure goes back to Dauchet and Tison [1]. In this theory confluence-related properties on ground terms are easily expressible. An extension of the theory to multiple rewrite systems, as well as the decision procedure, has been formalized in Isabelle/HOL [2–4]. The code generation facilities of Isabelle then give rise to the certifier FORTify which checks certificate constructed by FORT-h [6]. FORTify takes as input an answer (YES/NO), a formula, a list of TRSs, and a certificate proving that the formula holds (does not hold) for the given TRSs. It then checks the integrity and validity of the certificate. A command-line version of the tool can be downloaded from

[https://fortissimo.uibk.ac.at/fort\(ify\)/](https://fortissimo.uibk.ac.at/fort(ify)/)

We refer to [5] for a detailed description of FORTify.

This year FORTify participates, together with FORT-h, in the TRS category of CoCo 2025.

## References

- [1] Max Dauchet and Sophie Tison. The Theory of Ground Rewrite Systems is Decidable. In *Proc. 5th IEEE Symposium on Logic in Computer Science*, pages 242–248, 1990. doi: [10.1109/LICS.1990.113750](https://doi.org/10.1109/LICS.1990.113750).
- [2] Alexander Lochmann. Reducing Rewrite Properties to Properties on Ground Terms, 2022. [https://isa-afp.org/entries/Rewrite\\_Properties\\_Reduction.html](https://isa-afp.org/entries/Rewrite_Properties_Reduction.html), Formal proof development.
- [3] Alexander Lochmann and Bertram Felgenhauer. First-Order Theory of Rewriting. *Archive of Formal Proofs*, 2022. [https://isa-afp.org/entries/F0\\_Theory\\_Rewriting.html](https://isa-afp.org/entries/F0_Theory_Rewriting.html), Formal proof development.
- [4] Alexander Lochmann, Bertram Felgenhauer, Christian Sternagel, René Thiemann, and Thomas Sternagel. Regular Tree Relations. *Archive of Formal Proofs*, 2021. [https://www.isa-afp.org/entries/Regular\\_Tree\\_Relations.html](https://www.isa-afp.org/entries/Regular_Tree_Relations.html), Formal proof development.
- [5] Aart Middeldorp, Alexander Lochmann, and Fabian Mitterwallner. First-Order Theory of Rewriting for Linear Variable-Separated Rewrite Systems: Automation, Formalization, Certification. *Journal of Automated Reasoning*, 67(14):1–76, 2023. doi: [10.1007/s10817-023-09661-7](https://doi.org/10.1007/s10817-023-09661-7).
- [6] Fabian Mitterwallner and Aart Middeldorp. CoCo 2025 Participant: FORT-h 2.1. In *Proc. 14th International Workshop on Confluence*, 2025. This volume.

# The System SOL version 2025

Makoto Hamana, Shotaro Karasaki

Kyushu Institute of Technology, Japan  
hamana@csn.kyutech.ac.jp    karasaki.asop@gmail.com

SOL is a Haskell-based tool for showing confluence and strong normalisation of higher-order computation. SOL is intended to be a generic higher-order computation analysis tool that is applicable to the modern theories of higher-order programming languages. This aim is demonstrated in [Ham19] and further developed in [HAK20].

Based on the foundation of second-order algebraic theories [FH10] and its computational counter part [Ham19] and polymorphic extension [HAK20], we have implemented various results on higher-order syntax and computation in SOL, including Knuth and Bendix’s critical pair checking for confluence.

Recently, Muroya and Hamana have proposed a framework referred to as Term Evaluation Systems (TERS), unifying operational semantics and refinement reasoning [MH24]. We have also implemented contextual improvement verification by critical pair analysis [MH24] in SOL. Since every context-sensitive rewriting system can be simulated by a nondeterministic term evaluation system, in the present SOL 2025 version, we have implemented confluence checking of context-sensitive rewriting on an experimental basis.

## References

- [MH24] Koko Muroya and M. Hamana. Term Evaluation Systems with Refinements: First-Order, Second-Order, and Contextual Improvement, *Proc. of 17th International Symposium on Functional and Logic Programming (FLOPS 2024)*, Lecture Notes in Computer Science 14659, pp. 31-61, Springer, 2024.
- [FH10] M. Fiore and C.-K. Hur. Second-order equational logic. In *Proc. of CSL’10*, LNCS 6247, pages 320–335, 2010.
- [Ham19] M. Hamana. How to prove decidability of equational theories with second-order computation analyser SOL. *Journal of Functional Programming*, Cambridge University Press, Vol. 29, e20, 2019.
- [HAK20] Makoto Hamana, Tatsuya Abe, and Kentaro Kikuchi. Polymorphic Computation Systems: Theory and Practice of Confluence with Call-by-value, *Science of Computer Programming*, Elsevier, Volume 187, 102322, 15 February 2020.