Proceedings of the

11th International Workshop on Confluence

August 1, 2022

Haifa, Israel

Foreword

The 11th International Workshop on Confluence (IWC 2022) is held on August 1, 2022, in Haifa. IWC 2022 is affiliated with the 7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022), which is part of the Federated Logic Conference 2022 (FLoC 2022).

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 2022 continues this tradition. The present report comprises nine regular submissions and the abstract of an invited talk, as well as a summary of the 11th Confluence Competition (CoCo 2022). In the invited talk, Nao Hirokawa highlights the impressive progress in confluence research since the initial work by Knuth & Bendix and Huet, discussing different approaches to proving confluence that have advanced the area in the last 20 years. The regular contributions in these proceedings reflect the wide scope of current confluence research, 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 is devoted to CoCo 2022, providing a general overview as well as system descriptions of all competition entrants.

IWC 2022 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, subreviewers, and the organizers of CoCo. We are very grateful for their hard work. In addition, we want to thank the organizers of FSCD and FLoC for staging this exciting event that IWC 2022 can be part of.

Sarah Winkler and Camilo Rocha — Bolzano and Cali, 11 July 2022

Organization

IWC Steering Committee

- Takahito Aoto
- Mauricio Ayala-Rincón

IWC Program Committee

- Alejandro Díaz-Caro, Universidad Nacional de Quilmes, Argentina
- Francisco Durán, Universidad de Málaga, Spain
- Claudia Faggian, Université de Paris, France
- Raúl Gutiérrez, Universidad Politécnica de Madrid, Spain
- Makoto Hamana, Gunma University, Japan
- Naoki Nishida, Nagoya University, Japan
- Camilo Rocha, Pontificia Universidad Javeriana Cali, Colombia (co-chair)
- Sarah Winkler, Free University of Bozen-Bolzano, Italy (co-chair)

Additional Reviewers

- Pablo Barenbaum, Universidad de Buenos Aires, Argentina
- Pablo E. Martínez López, Universidad Nacional de Quilmes, Argentina

CoCo Steering Committee

- Raúl Gutiérrez, Universidad Politécnica de Madrid, Spain
- Aart Middeldorp, University of Innsbruck, Austria
- Naoki Nishida, Nagoya University, Japan
- Kiraku Shintani, JAIST, Japan

Contents

Seven Confluence Criteria for Solving COPS #20

Nao Hirokawa*

JAIST, Japan
hirokawa@jaist.ac.jp

COPS #20 is a thought-provoking confluence problem for term rewriting, posed by Gramlich and Lucas [2, Example 2].[1] Although the term rewrite system of the problem is confluent, it is beyond the realm of classical confluence criteria such as Knuth and Bendix' criterion [6] and Huet's parallel closedness [5]. In this talk we will discuss various solutions to the problem, recalling powerful confluence methods developed in the last decade and a half [1, 2, 3, 4, 7, 8, 9, 10].

References

[1] T. Aoto, J. Yoshida, and Y. Toyama. Proving confluence of term rewriting systems automatically. In *Proc. 20th International Conference on Rewriting Techniques and Applications*, volume 5595 of *LNCS*, pages 93–102, 2009.

[2] B. Gramlich and S. Lucas. Generalizing newman's lemma for left-linear rewrite systems. In *Proc. 17th International Conference on Rewriting Techniques and Applications*, volume 4098 of *LNCS*, pages 66–80, 2006.

[3] N. Hirokawa and A. Middeldorp. Decreasing diagrams and relative termination. *Journal of Automated Reasoning*, 47:481–501, 2011.

[4] N. Hirokawa, J. Nagele, V. van Oostrom, and M. Oyamaguchi. Confluence by critical pair analysis revisited. In *Proc. 27th International Conference on Automated Deduction*, volume 11716 of *LNCS*, pages 319–336, 2019.

[5] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[6] D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[7] J. Nagele, B. Felgenhauer, and A. Middeldorp. Improving automatic confluence analysis of rewrite systems by redundant rules. In *Proc. 26th International Conference on Rewriting Techniques and Applications*, volume 36 of *LIPIcs*, pages 257–268, 2015.

[8] K. Shintani and N. Hirokawa. Compositional confluence criteria. In *Proc. 7th International Conference on Formal Structures for Computation and Deduction*, volume 228 of *LIPIcs*, pages 28:1–28:19, 2022.

[9] Y. Toyama. Confluence criteria based on parallel critical pair closing, March 2017. Presented at the 46th TRS Meeting: https://www.trs.cm.is.nagoya-u.ac.jp/event/46thTRSmeeting/.

[10] V. van Oostrom. Confluence by decreasing diagrams, converted. In *Proc. 19th International Conference on Rewriting Techniques and Applications*, volume 5117 of *LNCS*, pages 306–320, 2008.

*Supported by JSPS KAKENHI Grant Number JP22K11900.

[1] https://cops.uibk.ac.at/?q=20

Development Closed Critical Pairs: Towards a Formalized Proof

Christina Kohl[1] and Aart Middeldorp[1]

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

Abstract

Having development closed critical pairs is a well-known sufficient condition for confluence of left-linear term rewrite systems. We present formalized results involving proof terms and unification that play an important role in the proof.

1 Introduction

In recent years several confluence criteria for first-order rewrite systems have been formalized in a proof assistant [4–6]. A well-known condition that has eluded all attempts so far is the result by van Oostrom [9] that a left-linear rewrite system is confluent if its critical pairs are development closed. In [2] it is suggested to use proof terms [8, Chapter 8] to obtain a rigorous proof. In [4] it is further suggested that a formalization of residual theory might be helpful. Here we pursue these suggestions further and present formalizations of several results that we believe will lead to a formal proof of the development closedness condition.

Our formalization is based on IsaFoR[1] and uses the existing formalizations of unification and critical pairs described in [7]. Our own development can be found at http://informatik-protem.uibk.ac.at/DC. To help readers negotiate the theory files we have annotated important results in this paper by a ☑-symbol which directly links to the HTML presentation of the corresponding result in our formalization.

2 Proof Terms

We use Greek letters for rule symbols which are used in proof terms. If α is a rule symbol then $\mathsf{lhs}(\alpha)$ ($\mathsf{rhs}(\alpha)$) denotes the left-hand (right-hand) side of the rewrite rule denoted by α. Furthermore $\mathsf{var}(\alpha)$ denotes the list $(x_1, \ldots, x_n)$ of variables appearing in α in some fixed order. The length of this list is the arity of α. The list $\mathsf{varpos}(\alpha) = (p_1, \ldots, p_n)$ denotes the corresponding variable positions in $\mathsf{lhs}(\alpha)$ such that $\mathsf{lhs}(\alpha)|_{p_i} = x_i$. Given a rule symbol α with $\mathsf{var}(\alpha) = (x_1, \ldots, x_n)$ and terms $t_1, \ldots, t_n$, we write $\langle t_1, \ldots, t_n \rangle_\alpha$ for the substitution $\{x_i \mapsto t_i \mid 1 \leqslant i \leqslant n\}$. Given a proof term A, its source $\mathsf{src}(A)$ and target $\mathsf{tgt}(A)$ are computed by the following equations for $\mathsf{st} \in \{\mathsf{src}, \mathsf{tgt}\}$:

$$
\begin{aligned}
\mathsf{st}(x) &= x \\
\mathsf{st}(f(A_1, \ldots, A_n)) &= f(\mathsf{st}(A_1), \ldots, \mathsf{st}(A_n)) \\
\mathsf{src}(\alpha(A_1, \ldots, A_n)) &= \mathsf{lhs}(\alpha)\langle \mathsf{src}(A_1), \ldots, \mathsf{src}(A_n) \rangle_\alpha \\
\mathsf{tgt}(\alpha(A_1, \ldots, A_n)) &= \mathsf{rhs}(\alpha)\langle \mathsf{tgt}(A_1), \ldots, \mathsf{tgt}(A_n) \rangle_\alpha
\end{aligned}
$$

Proof terms A and B are said to be *co-initial* if they have the same source. The proof term A can be seen as a witness of the multi-step $\mathsf{src}(A) \mathrel{\circ\!\!\!\!\longrightarrow} \mathsf{tgt}(A)$.For every multi-step there exists

[1] http://cl-informatik.uibk.ac.at/isafor

a proof term witnessing it. In the setting of left-linear TRSs we can extend the definition of src to contexts of proof terms by adding the clause $\mathsf{src}(\Box) = \Box$. Doing the same for tgt or for arbitrary TRSs however could lead to more than one hole appearing in the result. The following result is an easy consequence of the idempotence of src and tgt.

Lemma 1. *For any substitution σ, proof term context C, and proof term A we have*

$$\mathsf{src}(A\sigma) = \mathsf{src}(\mathsf{src}(A)\sigma) \qquad \mathsf{tgt}(A\sigma) = \mathsf{tgt}(\mathsf{tgt}(A)\sigma)$$
$$\mathsf{src}(C[A]) = \mathsf{src}(C[\mathsf{src}(A)]) = \mathsf{src}(C)[\mathsf{src}(A)]) \qquad \mathsf{tgt}(C[A]) = \mathsf{tgt}(C[\mathsf{tgt}(A)])$$

For co-initial proof terms A and B we can define partial operations *residual* (/), *join* ($\sqcup$), and *deletion* ($-$). The residual $A \,/\, B$ is used to compute which redexes in A remain after contracting the redexes of B, $A \sqcup B$ is used to obtain a single proof term containing all redexes of A and B, and $A - B$ is used to delete the redexes of B from A. The definitions can be found in Appendix A. Straightforward induction proofs on the definitions yield the following result.

Lemma 2. *1. If $A \,/\, B$ and $B \,/\, A$ are defined then $\mathsf{src}(B \,/\, A) = \mathsf{tgt}(A)$ and $\mathsf{tgt}(A \,/\, B) = \mathsf{tgt}(B \,/\, A)$.*

2. If $A \star B$ is defined then $\mathsf{src}(A \star B) = \mathsf{src}(A) = \mathsf{src}(B)$ for $\star \in \{\sqcup, -\}$.

The rules below can be used to compute joins, residuals, and deletions if the proof terms involved adhere to certain patterns.

Lemma 3. *Let $\star \in \{\sqcup, /, -\}$.*

1. *$A \star \mathsf{src}(A) = A$*
2. *If $A \star B = D$ then $C[A] \star \mathsf{src}(C)[B] = C[D]$ for any proof term context C.*
3. *If $\sigma(x) = \mathsf{src}(\tau(x))$ for all $x \in \mathcal{V}\mathsf{ar}(A)$ then $A\sigma \sqcup \mathsf{src}(A)\tau = A\tau$.*

Since the residual and deletion operations are not symmetric (as opposed to join) there is no obvious extension of the last item to / and $-$.

3 Development Closed Critical Pairs

To show that a left-linear TRS is confluent if it is development closed it suffices to show that $\twoheadrightarrow\!\!\!\!\!\circ$ has the diamond property. A sketch of this proof is depicted in Figure 3. There the multi-step $s \twoheadrightarrow\!\!\!\!\!\circ\; t$ is witnessed by the proof term A, the multi-step $s \twoheadrightarrow\!\!\!\!\!\circ\; u$ is witnessed by the proof term B, and we need to show $t \twoheadrightarrow\!\!\!\!\!\circ\; v \twoheadleftarrow\!\!\!\!\!\circ\; u$ for some term v. The idea is to use well-founded induction on the amount of overlap between A and B. The case where A and B do not overlap is straightforward since then the proof terms $A \,/\, B$ and $B \,/\, A$ are well-defined and have the same target (Lemma 2). In the other case we can use the fact that the TRS is development closed to show that the co-initial proof terms $A \,/\, \Delta_1$ and $D \sqcup (B - \Delta_2) \,/\, \Delta_1$ can be constructed and that the overlap between these is less than between A and B. A key ingredient for the proof is the notion of an *innermost overlap* between A and B. Here an overlap is simply a pair of positions (p, q) such that the redex in A at position p overlaps with the redex in B at position q. An innermost overlap is one where no other overlap occurs below it. Formal definitions can be found in Appendix C.

Assuming that A and B have overlap, we select an innermost overlap (p, q) and assume $q \leqslant p$ without loss of generality. Now let $q' = p \backslash q$, $\mathsf{varpos}(\mathsf{lhs}(\alpha)) = (p_1, \ldots, p_n)$, $\mathsf{var}(\mathsf{lhs}(\alpha)) =$

$(x_1, \dots, x_n)$, $\mathsf{varpos}(\mathsf{lhs}(\beta)) = (q_1, \dots, q_m)$, and $\mathsf{var}(\mathsf{lhs}(\beta)) = (y_1, \dots, y_m)$. We assume without loss of generality $\mathcal{V}\mathsf{ar}(\mathsf{lhs}(\alpha)) \cap \mathcal{V}\mathsf{ar}(\mathsf{lhs}(\beta)) = \varnothing$ and define a substitution σ that maps the variables of $\mathsf{lhs}(\alpha)$ and $\mathsf{lhs}(\beta)$ to subterms of s such that $\mathsf{lhs}(\alpha)\sigma = s|_p$ and $\mathsf{lhs}(\beta)\sigma = s|_q$:

$$\sigma = \{\, x_i \mapsto s|_{pp_i} \mid 1 \leqslant i \leqslant n \,\} \cup \{\, y_j \mapsto s|_{qq_j} \mid 1 \leqslant j \leqslant m \,\}$$

Furthermore we can use Lemma 9 of Appendix B to obtain another substitution τ which is an mgu of $\mathsf{lhs}(\alpha)$ and $\mathsf{lhs}(\beta)|_{q'}$:

$$\begin{aligned}\tau = &\{\, x_i \mapsto \mathsf{lhs}(\beta)|_{q'p_i} \mid 1 \leqslant i \leqslant n \text{ and } q'p_i \in \mathcal{P}\mathsf{os}(\mathsf{lhs}(\beta)) \,\} \cup \\ &\{\, y_j \mapsto \mathsf{lhs}(\alpha)|_{q_j \backslash q'} \mid 1 \leqslant j \leqslant m \text{ and } q_j \backslash q' \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(\mathsf{lhs}(\alpha)) \,\}\end{aligned}$$

Hence we obtain the critical peak

$$\mathsf{lhs}(\beta)[\mathsf{rhs}(\alpha)\tau]_{q'} \xleftarrow[\alpha]{q'} \mathsf{lhs}(\beta)[\mathsf{lhs}(\alpha)\tau]_{q'} = \mathsf{lhs}(\beta)\tau \xrightarrow[\beta]{\epsilon} \mathsf{rhs}(\beta)\tau$$

Assuming that the given TRS is development closed, we know there exists a multi-step $\mathsf{lhs}(\beta)[\mathsf{rhs}(\alpha)\tau]_{q'} \multimap\!\!\!\!\rightarrow \mathsf{rhs}(\beta)\tau$. Let D' be the proof term representation of such a multi-step and define $D = s[D'\sigma]_q$. Then we can prove the following result.

Lemma 4. *The proof term D witnesses the multi-step $\mathsf{tgt}(\Delta_1) \multimap\!\!\!\!\rightarrow \mathsf{tgt}(\Delta_2)$.*

Ultimately we need to also show that $D \sqcup (B - \Delta_2) \,/\, \Delta_1$ is well-defined and witnesses $\mathsf{tgt}(\Delta_1) \multimap\!\!\!\!\rightarrow \mathsf{tgt}(B)$. For this purpose we introduce another substitution ρ:

$$\rho = \{\, y_j \mapsto B_j \mid 1 \leqslant j \leqslant m \,\} \cup \{\, x_i \mapsto \mathsf{lhs}(\beta)\langle B_1, \dots, B_m\rangle_\beta|_{q'p_i} \mid 1 \leqslant i \leqslant n \,\}$$

Note the similarity to σ: ρ maps to subterms of B while σ maps to the sources of these proof terms. The key property that makes ρ useful for computing $(B - \Delta_2) \,/\, \Delta_1$ is the following:

Lemma 5. *If $1 \leqslant j \leqslant m$ then $\tau(y_j)\rho = B_j$.*

Proof. We distinguish two cases: $\tau(y_j) = y_j$ and $\tau(y_j) \neq y_j$. In the first case we immediately obtain $\tau(y_j)\rho = \rho(y_j) = B_j$ from the definition of ρ. For the second case first observe that if all function symbols of $\mathsf{lhs}(\alpha)$ also appear in $\mathsf{lhs}(\beta)\langle B_1, \dots, B_m\rangle_\beta|_{q'}$ (i.e., no rule symbols are in the way) then it follows from the definition of ρ that

$$\mathsf{lhs}(\alpha)\rho = \mathsf{lhs}(\beta)\langle B_1, \dots, B_m\rangle_\beta|_{q'} \tag{$*$}$$

Checking that all function symbols of $\mathsf{lhs}(\alpha)$ also appear in $\mathsf{lhs}(\beta)\langle B_1, \dots, B_m\rangle_\beta|_{q'}$ can be done by verifying

$$p' \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(\mathsf{lhs}(\alpha)) \quad\Longrightarrow\quad \mathsf{src}^\sharp(\mathsf{lhs}(\beta)\langle B_1, \dots, B_m\rangle_\beta|_{q'})(p') \text{ is unlabeled}$$

which relies on the fact that having a labeled function symbol at such a position p' would contradict the assumption that (p, q) is an innermost overlap of A and B. With $(*)$ we obtain $\tau(y_j)\rho = \mathsf{lhs}(\alpha)|_{q_j \backslash q'}\rho = (\mathsf{lhs}(\beta)\langle B_1, \dots, B_m\rangle_\beta|_{q'})|_{q_j \backslash q'} = \mathsf{lhs}(\beta)\langle B_1, \dots, B_m\rangle_\beta|_{q_j} = B_j$. ☐

The term $s = \mathsf{src}(A) = \mathsf{src}(B)$ contains a redex with respect to β at position q. In the following we denote by q_β the corresponding position of the rule symbol β in the proof term B, i.e., the position q_β such that $B = B[\beta(B_1, \dots, B_m)]_{q_\beta}$ and $\mathsf{src}(B)[\ ]_q = \mathsf{src}(B[\ ]_{q_\beta})$. It can be shown that such a position exists for arbitrary proof terms B and positions q where $\mathsf{src}^\sharp(B)(q) = \beta^0$.

Lemma 6. *1.* $D \sqcup (B - \Delta_2) \mathbin{/} \Delta_1 = B[D'\rho]_{q_\beta}$

2. $D \sqcup (B - \Delta_2) \mathbin{/} \Delta_1$ *witnesses* $\mathsf{tgt}(\Delta_1) \mathrel{\circ\!\!\!\to\!\!\!\!\to} \mathsf{tgt}(B)$

Proof. From Lemma 3 we obtain $B - \Delta_2 = B[\mathsf{lhs}(\beta)\langle B_1, \dots, B_m\rangle_\beta]_{q_\beta}$ and with Lemma 5 we further obtain $B[\mathsf{lhs}(\beta)\langle B_1, \dots, B_m\rangle_\beta]_{q_\beta} = B[\mathsf{lhs}(\beta)\tau\rho]_{q_\beta} = B[\mathsf{lhs}(\beta)[\mathsf{lhs}(\alpha)\tau]_{q'}\rho]_{q_\beta}$. Another application of Lemma 3 yields $(B - \Delta_2) \mathbin{/} \Delta_1 = B[\mathsf{lhs}(\beta)[\mathsf{rhs}(\alpha)\tau]_{q'}\rho]_{q_\beta}$. From Lemma 3(3) we obtain $D'\sigma \sqcup \mathsf{lhs}(\beta)[\mathsf{rhs}(\alpha)\tau]_{q'}\rho = D'\rho$ and since $\mathsf{src}(B[\,]_{q_\beta}) = s[\,]_q$ and $D = s[D'\sigma]_q$ we can apply Lemma 3(2) (modulo symmetry of $\sqcup$) to obtain the desired $D \sqcup (B - \Delta_2) \mathbin{/} \Delta_1 = B[D'\rho]_{q_\beta}$. From Lemma 2 and Lemma 4 we obtain $\mathsf{src}((B - \Delta_2) \mathbin{/} \Delta_1) = \mathsf{tgt}(\Delta_1) = \mathsf{src}(D)$ and hence $\mathsf{src}(D \sqcup (B - \Delta_2) \mathbin{/} \Delta_1) = \mathsf{tgt}(\Delta_1)$. It remains to show that $\mathsf{tgt}(D \sqcup (B - \Delta_2) \mathbin{/} \Delta_1) = \mathsf{tgt}(B)$. We have

$$\begin{aligned}
\mathsf{tgt}(D \sqcup (B - \Delta_2) \mathbin{/} \Delta_1) &= \mathsf{tgt}(B[D'\rho]_{q_\beta}) \\
&= \mathsf{tgt}(B[\mathsf{tgt}(\mathsf{rhs}(\beta)\tau\rho)]_{q_\beta}) && \text{(Lemma 1 and definition of } D'\text{)} \\
&= \mathsf{tgt}(B[\mathsf{tgt}(\mathsf{rhs}(\beta)\langle B_1, \dots, B_m\rangle_\beta)]_{q_\beta}) && \text{(Lemma 5)} \\
&= \mathsf{tgt}(B[\beta(B_1, \dots, B_m)]_{q_\beta}) && \text{(Lemma 1)} \\
&= \mathsf{tgt}(B) && \square
\end{aligned}$$

In order to apply the induction hypothesis and to conclude the proof in Figure 3 it remains to show that the amount of overlap between the proof terms $A \mathbin{/} \Delta_1$ and $D \sqcup (B - \Delta_2) \mathbin{/} \Delta_1$ is less than the amount of overlap between A and B. Like the proof of Lemma 5 this relies on the fact that we chose an innermost overlap (p, q) during the construction of D. At the time of writing the formalization of this fact is still work in progress.

The example below illustrates the constructions of Lemma 6 for specific proof terms A and B. It can be retraced in the tool ProTeM [3] where we implemented all important operations.

Example 7. Consider the left-linear and development closed TRS $\mathcal{R}$ consisting of the rules

$$\alpha\colon \mathsf{f}(x_1, \mathsf{g}(x_2)) \to \mathsf{f}(x_1, \mathsf{g}(x_1)) \quad \beta\colon \mathsf{f}(\mathsf{g}(y_1), y_2) \to \mathsf{f}(\mathsf{g}(y_1), \mathsf{g}(y_1)) \quad \gamma\colon \mathsf{g}(\mathsf{a}) \to \mathsf{g}(\mathsf{b}) \quad \delta\colon \mathsf{b} \to \mathsf{a}$$

and the proof terms $A = \mathsf{g}(\alpha(\gamma, \mathsf{a}))$ and $B = \mathsf{g}(\beta(\mathsf{a}, \gamma))$. We have $\mathsf{src}(A) = \mathsf{src}(B) = \mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a})))$ and $\mathsf{overlaps}(A, B) = \{(1, 1), (1, 12), (11, 1)\}$ where both the second and third overlap are innermost. For the overlap $(11, 1)$ we obtain the substitution $\tau = \{y_1 \mapsto \mathsf{a}\}$ with corresponding critical peak

$$\mathsf{f}(\mathsf{g}(\mathsf{b}), y_2) \xleftarrow[\gamma]{1} \mathsf{f}(\mathsf{g}(\mathsf{a}), y_2) \xrightarrow[\beta]{\epsilon} \mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a}))$$

This critical peak can be closed by applying β at the root and δ at position 11 in the term $\mathsf{f}(\mathsf{g}(\mathsf{b}), y_2)$ as witnessed by the proof term $D' = \beta(\delta, y_2)$. Since $\sigma = \{y_1 \mapsto \mathsf{a},\ y_2 \mapsto \mathsf{g}(\mathsf{a})\}$ we have $D = s[D'\sigma]_1 = s[\beta(\delta, \mathsf{g}(\mathsf{a}))]_1 = \mathsf{g}(\beta(\delta, \mathsf{g}(\mathsf{a})))$. Furthermore, $\Delta_1 = \mathsf{g}(\mathsf{f}(\gamma, \mathsf{g}(\mathsf{a})))$, $\Delta_2 = \mathsf{g}(\beta(\mathsf{a}, \mathsf{g}(\mathsf{a})))$, $\rho = \{y_1 \mapsto \mathsf{a},\ y_2 \mapsto \gamma\}$ and hence

$$\begin{aligned}
B - \Delta_2 &= \mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{a}), \gamma)) = B[\mathsf{lhs}(\beta)\tau\rho]_1 \\
(B - \Delta_2) \mathbin{/} \Delta_1 &= \mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{b}), \gamma)) = B[\mathsf{lhs}(\beta)[\mathsf{rhs}(\gamma)\tau]_1\rho]_1 \\
D \sqcup (B - \Delta_2) \mathbin{/} \Delta_1 &= \mathsf{g}(\beta(\delta, \gamma)) \quad = B[D'\rho]_1
\end{aligned}$$

For the non-innermost overlap $(1, 1)$ the term $(B - \Delta_2) \mathbin{/} \Delta_1$ as well as the substitution ρ are not well-defined. We have $\Delta_1 = \mathsf{g}(\alpha(\mathsf{g}(\mathsf{a}), \mathsf{a}))$ and $\Delta_2 = \mathsf{g}(\beta(\mathsf{a}, \mathsf{g}(\mathsf{a})))$ and hence $B - \Delta_2 = s[(\mathsf{f}(\mathsf{g}(\mathsf{a}), \gamma))]_1$. Since $\mathsf{f}(\mathsf{g}(\mathsf{a}), \gamma)$ does not match $\mathsf{lhs}(\alpha)$ the result of $(B - \Delta_2) \mathbin{/} \Delta_1$ is undefined. Also the substitution ρ cannot be computed since the variable binding $x_2 \mapsto \mathsf{lhs}(\beta)\langle B_1, \dots, B_m\rangle_\beta|_{21} = \mathsf{f}(\mathsf{g}(\mathsf{a}), \gamma)|_{21}$ does not make sense.

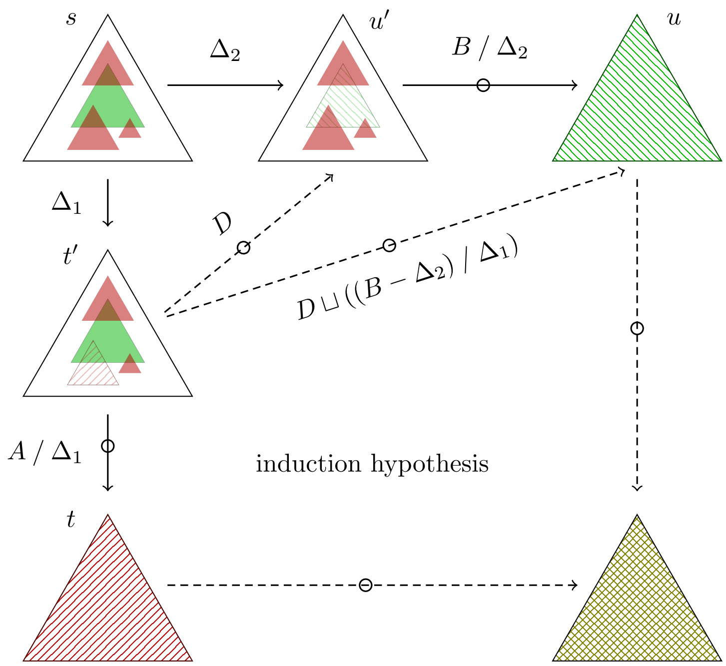


Figure 1: Picture proof.

References

[1] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. doi:10.1017/CBO9781139172752.

[2] Nao Hirokawa and Aart Middeldorp. Commutation via relative termination. In *Proc. 2th IWC*, pages 29–33, 2013.

[3] Christina Kohl and Aart Middeldorp. ProTeM: A proof term manipulator (system description). In *Proc. 3rd FSCD*, volume 108 of *LIPIcs*, pages 31:1–31:8, 2018. doi:10.4230/LIPIcs.FSCD.2018.31.

[4] Julian Nagele and Aart Middeldorp. Certification of classical confluence results for left-linear term rewrite systems. In *Proc. 7th ITP*, volume 9807 of *LNCS*, pages 290–306, 2016. doi:10.1007/978-3-319-43144-4_18.

[5] Julian Nagele and Harald Zankl. Certified rule labeling. In *Proc. 26th RTA*, volume 36 of *LIPIcs*, pages 269–284, 2015. doi:10.4230/LIPIcs.RTA.2015.269.

[6] Ana Cristina Rocha-Oliveira, André Luiz Galdino, and Mauricio Ayala-Rincón. Confluence of orthogonal term rewriting systems in the prototype verification system. *Journal of Automated Reasoning*, 58(2):231–251, 2017. doi:10.1007/s10817-016-9376-2.

[7] Christian Sternagel and René Thiemann. Formalizing Knuth–Bendix orders and Knuth–Bendix completion. In *Proc. 23rd RTA*, volume 21 of *LIPIcs*, pages 287–302, 2013. doi:10.4230/LIPIcs.RTA.2013.287.

[8] TeReSe, editor. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[9] Vincent van Oostrom. Developing developments. *Theoretical Computer Science*, 175(1):159–181, 1997. doi:10.1016/S0304-3975(96)00173-9.

On local confluence of conditional rewrite systems[*]

Salvador Lucas

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

Abstract

We characterize local confluence of *conditional rewrite systems* as the joinability of a set of conditional pairs including the usual conditional critical pairs and a new kind of pairs we call *conditional variable pairs*.

1 Introduction

In a landmark 1980 paper, Gérard Huet proved the following result for Term Rewriting Systems (TRSs [2]), see [6, Lemma 3.1] (we borrow [2, Theorem 6.2.4] here):

> A TRS $\mathcal{R}$ is *locally confluent* if and only if all its *critical pairs* are *joinable*.

A critical pair $\langle s,t\rangle$ is easily obtained from *rules* $\ell \to r$ and $\ell' \to r'$ of $\mathcal{R}$ by *unification* between a non-variable subterm of ℓ and the whole ℓ'. Pairs $\langle s,t\rangle$ are *joinable* if both s and t can be rewritten to the same term u in zero or more steps. Huet's result provides a straightforward criterion for *disproving* confluence of TRSs $\mathcal{R}$: the existence of *non-joinable* critical pairs implies that $\mathcal{R}$ is *not* locally confluent nor confluent. Together with Newman's Lemma, it yields a characterization of *confluence* for terminating TRSs [2, Corollary 6.2.5].

Conditional TRSs (CTRSs, see, e.g., [9, Chapter 7]) consist of rules $\ell \to r \Leftarrow c$ where a sequence c of conditions $s \approx t$ controls the application of a rewriting step with $\ell \to r$. The notion of *conditional critical pair* $\langle s,t\rangle \Leftarrow c$ exists for CTRSs, and a modified notion of joinability is introduced to take into account the conditional part c [7, 4, 1]), in sharp contrast to TRSs, though, there are non-locally confluent CTRSs *without* conditional critical pairs.

Example 1. *The following (oriented) CTRS $\mathcal{R}$ [9, Example 7.3.3]:*

$$\mathsf{a} \quad \to \quad \mathsf{b} \tag{1}$$

$$\mathsf{f}(x) \quad \to \quad \mathsf{c} \Leftarrow x \approx \mathsf{a} \tag{2}$$

has no conditional critical pair. However, we have $\mathsf{f}(\underline{\mathsf{a}}) \to_{(1)} \mathsf{f}(\mathsf{b})$ *and* $\underline{\mathsf{f}(\mathsf{a})} \to_{(2)} \mathsf{c}$ *(this is because, when variable x in the left-hand side $\mathsf{f}(x)$ of rule (2) is instantiated to a, the corresponding instance $\mathsf{a} \approx \mathsf{a}$ of the condition $x \approx \mathsf{a}$ of the rule is trivially satisfied). Thus, we obtain a* peak

$$\mathsf{f}(\mathsf{b}) \;{}_{(1)}\!\leftarrow \quad \mathsf{f}(\mathsf{a}) \quad \to_{(2)} \mathsf{c} \tag{3}$$

Since $\mathsf{f}(\mathsf{b})$ and c are irreducible, *they are* not *joinable and $\mathcal{R}$ is not (locally) confluent.*

In this paper we show that there is a set of *conditional pairs* (properly extending the set of conditional critical pairs) whose joinability *characterizes* local confluence of CTRSs. For instance, for $\mathcal{R}$ in Example 1, the following pair witnesses non-confluence of $\mathcal{R}$:

$$\langle \mathsf{f}(x'), \mathsf{c}\rangle \Leftarrow x \to x', x \approx \mathsf{a} \tag{4}$$

where x and x' are distinct variables. This is an example of a *conditional variable pair*, a new class of conditional pairs that we introduce here.

[*]Partially supported by MCIN/AEI project RTI2018-094403-B-C32 and GV project PROMETEO/2019/098.

2 First-order theory of a CTRS. Rewriting as deduction

As explained in [8, Section 4.5], given a CTRS $\mathcal{R}$ both $\to_{\mathcal{R}}$ and $\to^*_{\mathcal{R}}$ are defined as deducibility of goals $s \to t$ and $s \to^* t$ (where $\to$ and $\to^*$ are viewed as predicate symbols) in a first-order theory $\overline{\mathcal{R}}$ associated to $\mathcal{R}$ consisting of the following components: (i) a sentence

$$(\forall x)\ x \to^* x \tag{5}$$

expressing *reflexivity* of many-step rewriting; (ii) a sentence

$$(\forall x, y, z)\ x \to y \wedge y \to^* z \Rightarrow x \to^* z \tag{6}$$

expressing *compatibility* of one-step and many-step rewriting; (iii) for each k-ary function symbol f and $1 \leq i \leq k$, a sentence

$$(\forall x_1, \ldots, x_i, \ldots, x_k, y_i)\ x_i \to y_i \Rightarrow f(x_1, \ldots, x_i, \ldots, x_k) \to f(x_1, \ldots, y_i, \ldots, x_k) \tag{7}$$

where $x_1, \ldots, x_k$ and y_i are distinct variables, enabling the *propagation* of rewriting steps inside other terms; and (iv) for each rule $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$ in $\mathcal{R}$ a sentence

$$(\forall x_1, \ldots, x_n)\ s_1 \approx t_1 \wedge \cdots \wedge s_n \approx t_n \Rightarrow \ell \to r \tag{8}$$

where $x_1, \ldots, x_n$ are the variables occurring in the rule. Sentence (8) expresses the possibility of applying a rewriting step $\sigma(\ell) \to \sigma(r)$ to for some substitution σ provided that, for all $1 \leq i \leq n$, $\sigma(s_i) \approx \sigma(t_i)$ can be proved. Besides, (v) we need to define the *meaning* of predicate $\approx$ used in the conditions. In this note, we exemplify our treatment for *oriented* CTRSs, where instantiated conditions $\sigma(s_i) \approx \sigma(t_i)$ are treated as *reachability tests* $\sigma(s_i) \to^* \sigma(t_i)$. Thus, the following sentence should be added:

$$(\forall x, y) \qquad x \to^* y \Rightarrow x \approx y \tag{9}$$

Example 2. *For $\mathcal{R}$ in Example 1, $\overline{\mathcal{R}} = \{(5), (6), (10), (11), (12), (9)\}$ with*

$$(\forall x, y) \qquad x \to y \Rightarrow \mathsf{f}(x) \to \mathsf{f}(y) \tag{10}$$

$$\mathsf{a} \to \mathsf{b} \tag{11}$$

$$(\forall x) \qquad x \approx \mathsf{a} \Rightarrow \mathsf{f}(x) \to \mathsf{c} \tag{12}$$

Example 3. *For the following oriented CTRS $\mathcal{R}$:*

$$\mathsf{b} \to \mathsf{c} \tag{13}$$

$$\mathsf{g}(\mathsf{d}) \to \mathsf{b} \tag{14}$$

$$\mathsf{g}(x) \to \mathsf{c} \Leftarrow \mathsf{g}(x) \approx \mathsf{b} \tag{15}$$

we have $\overline{\mathcal{R}} = \{(5), (6), (16), (17), (18), (19), (9)\}$ with

$$(\forall x, y) \qquad x \to y \Rightarrow \mathsf{g}(x) \to \mathsf{g}(y) \tag{16}$$

$$\mathsf{b} \to \mathsf{c} \tag{17}$$

$$\mathsf{g}(\mathsf{d}) \to \mathsf{b} \tag{18}$$

$$(\forall x) \qquad \mathsf{g}(x) \approx \mathsf{b} \Rightarrow \mathsf{g}(x) \to \mathsf{c} \tag{19}$$

For all terms s and t we write $s \to_{\mathcal{R}} t$ (resp. $s \to^*_{\mathcal{R}} t$) iff $\overline{\mathcal{R}} \vdash s \to t$ (resp. $\overline{\mathcal{R}} \vdash s \to^* t$) holds. As usual, "*iff*" means "*if and only if*" and '$\vdash$' denotes *deducibility*. Terms s and t are *joinable* if there is a term u such that both $s \to^*_{\mathcal{R}} u$ and $t \to^*_{\mathcal{R}} u$ hold. We say that $\mathcal{R}$ is (locally) confluent iff $\to_{\mathcal{R}}$ is (locally) confluent. Also, $\mathcal{R}$ is terminating iff $\to_{\mathcal{R}}$ is terminating.

3 Peaks and (local) confluence

Consider a CTRS $\mathcal{R}$, a term s, positions $\overline{p}, \overline{p'} \in \mathcal{P}os(s)$, rules $\alpha : \ell \to r \Leftarrow c$ and $\alpha' : \ell' \to r' \Leftarrow c'$, and substitutions σ and σ', such that (i) $s|_{\overline{p}} = \sigma(\ell)$ and $\sigma(c)$ holds in $\overline{\mathcal{R}}$, i.e., if c is $s_1 \approx t_1, \ldots, s_n \approx t_n$, then for all $1 \leq i \leq n$ $\overline{\mathcal{R}} \vdash \sigma(s_i) \approx \sigma(t_i)$ holds; and (ii) $s|_{\overline{p'}} = \sigma'(\ell')$ and $\sigma'(c')$ holds. The situation

$$\overline{u} = s[\sigma'(r')]_{\overline{p'}} \ {}_{\mathcal{R}}{\leftarrow}\ s[\sigma'(\ell')]_{\overline{p'}} \quad = \quad s \quad = \quad s[\sigma(\ell)]_{\overline{p}} \ \to_{\mathcal{R}}\ s[\sigma(r)]_{\overline{p}} = \overline{v} \tag{20}$$

is called a *peak*. A CTRS is locally confluent if and only if for all terms s, $\overline{u}$, $\overline{v}$ defining a peak (20), $\overline{u}$ and $\overline{v}$ are joinable. Depending on the relative location of positions $\overline{p}$ and $\overline{p'}$ in (20), different classes of peaks are usually distinguished [4, Sections 2.1–2.3]:

***Disjoint* peaks.** If $\overline{p}$ and $\overline{p'}$ in (20) are *disjoint*, i.e., $\overline{p} \parallel \overline{p'}$, then $s = s[\sigma(\ell)]_{\overline{p}}[\sigma'(\ell')]_{\overline{p'}} = s[\sigma'(\ell')]_{\overline{p'}}[\sigma(\ell)]_{\overline{p}}$. Accordingly, (20) can be written as follows:

$$\overline{u} = s[\sigma'(r')]_{\overline{p'}}[\sigma(\ell)]_{\overline{p}} = s[\sigma'(r')]_{\overline{p'}} \ {}_{\mathcal{R}}{\leftarrow} s[\sigma'(\ell')]_{\overline{p'}}[\sigma(\ell)]_{\overline{p}} \to_{\mathcal{R}} s[\sigma(r)]_{\overline{p}} = s[\sigma'(\ell')]_{\overline{p'}}[\sigma(r)]_{\overline{p}} = \overline{v} \tag{21}$$

Disjoint peaks (21) are always *joinable*.

***Critical* peaks.** If $\overline{p'} = \overline{p}.p \in \mathcal{P}os(s)$ for some non-variable position $p \in \mathcal{P}os_{\mathcal{F}}(\ell)$, then $s = s[\sigma(\ell)[\sigma'(\ell')]_p]_{\overline{p}}$, $\sigma(\ell) = \sigma(\ell)[\sigma'(\ell')]_p$, and after removing the untouched context around $\sigma(\ell)$, which remains unchanged in both rewriting steps of the peak, and assuming that α and α' share no variable (rename if necessary), we can use a single substitution σ to obtain the usual simpler form of (20) (which is then called a *critical peak*) as follows:

$$u = \sigma(\ell)[\sigma(r')]_p \ {}_{\mathcal{R}}{\leftarrow} \quad \sigma(\ell)[\sigma(\ell')]_p \quad \to_{\mathcal{R}} \ \sigma(r) = v \tag{22}$$

***Improper* critical peaks.** If (22) is obtained from a single rule $\alpha : \ell \to r \Leftarrow c$ which is used twice, i.e., α' in (22) is a renamed version of α, and $p = \Lambda$ is the *root* position, we call it an *improper critical peak*. Since $p = \Lambda$ and $s = \sigma(\ell) = \sigma(\ell')$, (22) becomes

$$\sigma(r') \ {}_{\mathcal{R}}{\leftarrow} \quad \sigma(\ell') = \sigma(\ell) \quad \to_{\mathcal{R}} \ \sigma(r) \tag{23}$$

***Variable* peaks.** If $\overline{p'} \geq \overline{p}.p$ for some variable position $p \in \mathcal{P}os_{\mathcal{X}}(\ell)$, then $s = s[\sigma(\ell)[C[\sigma'(\ell')]]_p]_{\overline{p}}$ for some context $C[\,]$ and $\sigma(x) = C[\sigma'(\ell')]$. Thus, (20) can be (equivalently) written as follows:

$$u = \sigma(\ell)[C[\sigma'(r')]]_p \ {}_{\mathcal{R}}{\leftarrow} \quad \sigma(\ell)[C[\sigma'(\ell')]]_p \quad \to_{\mathcal{R}} \ \sigma(r) = v \tag{24}$$

Dershowitz et al. call (24) a *variable peak*. Note that $\sigma(\ell) = \sigma(\ell)[C[\sigma'(\ell')]]_p$. In the realm of TRSs, variable peaks (24) are *always joinable*. However, as observed in [4, Section 2.3], this is *not* true for CTRSs. In particular, (3) is a variable peak which is not joinable.

Since every peak (20) is either a disjoint peak (21), which is always joinable, a (proper or improper) critical peak (22), or a variable peak (24), we have the following.

Proposition 4. *A CTRS is locally confluent iff all peaks (22) and (24) are joinable.*

Note that critical peaks (22) include *improper* critical peaks (23).

4 From peaks to extended critical pairs

In the following, we deal with *conditional pairs*, $\langle s,t\rangle \Leftarrow c$, where s and t are terms and c is a sequence of conditions $s \to t$ or $s \approx t$. A conditional pair $\langle s,t\rangle \Leftarrow c$ is *joinable* if for all substitutions σ, whenever $\sigma(c)$ holds in $\overline{\mathcal{R}}$, terms $\sigma(s)$ and $\sigma(t)$ are joinable. Methods for (dis)proving joinability of conditional pairs have been investigated in [5, Section 6]. In particular, $\overline{\mathcal{R}}$-*infeasible* conditional pairs $\langle s,t\rangle \Leftarrow c$, i.e., such that $\sigma(c)$ holds (in $\overline{\mathcal{R}}$) for *no substitution* σ, are trivially joinable.

Conditional critical pairs. Let $\mathcal{R}$ be a CTRS, $\alpha : \ell \to r \Leftarrow c$ and $\alpha' : \ell' \to r' \Leftarrow c'$ be rules of $\mathcal{R}$ sharing no variable (rename if necessary), and $p \in \mathcal{P}os_{\mathcal{F}}(\ell)$ be a nonvariable position of ℓ such that $\ell|_p$ and ℓ' unify with *mgu* θ. Then,

$$\langle \theta(\ell[r']_p), \theta(r)\rangle \Leftarrow \theta(c), \theta(c') \tag{25}$$

is a *conditional critical pair* of $\mathcal{R}$ [9, Definition 7.1.8(1)]. Each critical peak κ is represented by a conditional critical pair π; joinability of π implies that of κ. If α and α' in (27) are renamed versions of the *same* rule, the case $p = \Lambda$ is usually considered *improper* to obtain a conditional critical pair. Thus, following the literature, we call *proper* to those conditional critical pairs *not fitting* this case. As usual, the set of *proper conditional critical pairs* of $\mathcal{R}$ is $\mathsf{CCP}(\mathcal{R})$.

Example 5. *The CTRS $\mathcal{R}$ in Example 3 has a proper conditional critical pair*

$$\langle \mathsf{b}, \mathsf{c}\rangle \Leftarrow \mathsf{g}(\mathsf{d}) \approx \mathsf{b} \tag{26}$$

This pair is clearly joinable *as* $\underline{\mathsf{b}} \to_{(13)} \mathsf{c}$ *holds.*

Improper conditional critical pairs. As shown in, e.g., [1, Example 4.1.a], improper critical pairs can jeopardize (local) confluence of CTRSs $\mathcal{R}$. Let $\alpha : \ell \to r \Leftarrow c \in \mathcal{R}$ such that c is not empty, and $\alpha' : \ell' \to r' \Leftarrow c'$ a renamed version of α so that α and α' share no variable. Let θ be a most general unifier of ℓ and ℓ'. Then,

$$\langle \theta(r'), \theta(r)\rangle \Leftarrow \theta(c), \theta(c') \tag{27}$$

is an *improper* conditional critical pair (iCCP) of $\mathcal{R}$ [1, Definition 4.2]. The set of *improper conditional critical pairs* of $\mathcal{R}$ is $\mathsf{iCCP}(\mathcal{R})$. For 2-CTRS, where rules $\ell \to r \Leftarrow c$ satisfy $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell)$, improper critical pairs are trivially joinable and can be dismissed.

Example 6. *Both CTRSs in Examples 1 and 3 are 2-CTRS. Thus, improper conditional critical pairs are dismissed.*

Improper conditional critical pairs are often ‘forgotten’ when analyzing confluence of CTRSs $\mathcal{R}$. Most tools do not mention or compute them. The first definitions of what, following [9, Definition 7.1.8(1)], we call conditional critical pair today, i.e., [7, Definition 3.2] (speaking about *contextual critical pairs*, though) and [4, Definition 3] (just talking about *critical pairs*) did *not* distinguish between proper and improper pairs. Hence, their results showing confluence by joinability of conditional critical pairs, e.g., [7, Theorems 3.3 & 5.3] and [3, Section 3] should be used considering both proper and improper pairs. Avenhaus and Loría-Sáenz also talk about *critical pairs* but explicitly distinguish proper and improper. However, their notation $\mathsf{CP}(\mathcal{R})$ *excludes* improper pairs from the set of conditional critical pairs associated to $\mathcal{R}$ [1, Definition 4.2]. Such a notation is used in their results (e.g., [1, Theorem 4.1]), i.e., only proper conditional critical pairs are considered. Ohlebusch's book does *not* talk about proper or improper pairs; however the aforementioned notion of *conditional critical pair* (CCP) actually refers to *proper* conditional pairs, as can be concluded from [9, Definition 7.1.8(1)]. Recent works about confluence of CTRSs usually follow this definition, see, e.g., [10]. Thus, prospective readers of the literature must be careful about what notion of conditional critical pair is actually used in each considered result. Moreover, important notions like *orthogonal CTRS* (requiring left-linearity plus the absence of conditional critical pairs, see, e.g., [9, Definition 7.1.10]), depend on what a conditional critical pair is. For instance, $\mathcal{R} = \{0 + y \to y, \mathsf{s}(x) + y \to x + \mathsf{s}(y), \mathsf{f}(x, y) \to z \Leftarrow x + y \approx z + z'\}$ [1, Example 4.1(a)] is a 3-CTRS which is orthogonal according to [9] (there is no proper CCP) but not according to [4] (there is an improper CCP). As noticed in [1], $\mathcal{R}$ is not confluent due to such an improper (but harmful) CCP.

Conditional variable pairs. In [4, Section 3], Dershowitz et al. show that variable peaks (24) of CTRSs may *fail* to be joinable. No corresponding notion of *pair* is given, though. The following definition captures variable peaks of CTRSs as a new kind of conditional pairs.

Definition 7 (Conditional variable pair)**.** *Let $\mathcal{R}$ be a CTRS, $\ell \to r \Leftarrow c \in \mathcal{R}$, $x \in \mathcal{V}ar(\ell)$, $p \in \mathcal{P}os_x(\ell)$, and x' be a fresh variable. Then,*

$$\langle \ell[x']_p, r\rangle \Leftarrow x \to x', c \tag{28}$$

is a conditional variable pair *(*CVP*). Variable x is called the* critical variable *of the pair, and p is called the* critical position*.*

Each variable peak κ is represented by a conditional variable pair π; joinability of π implies that of κ. The set of *conditional variable pairs* of a CTRS $\mathcal{R}$ is $\mathsf{CVP}(\mathcal{R})$. Unconditional rules may determine conditional variable pairs, but they are always joinable, and then can be dismissed.

Example 8. *Rule (2) of $\mathcal{R}$ in Example 1 defines the conditional variable pair (4).*

Example 9. *The CTRS $\mathcal{R}$ in Example 3 has the following conditional variable pair:*

$$\langle \mathsf{g}(x'), \mathsf{c}\rangle \Leftarrow x \to x', \mathsf{g}(x) \approx \mathsf{b} \tag{29}$$

This pair is joinable because the conditional part $x \to x', \mathsf{g}(x) \approx \mathsf{b}$ is $\overline{\mathcal{R}}$-infeasible *(this can be automatically proved with* infChecker *http://zenon.dsic.upv.es/infChecker/).*

5 Characterization of local confluence of CTRSs

All aforementioned kinds of conditional pairs are collected into a single set.

Definition 10 (Extended conditional critical pairs)**.** *Let $\mathcal{R}$ be a CTRS. The set*

$$\mathsf{ECCP}(\mathcal{R}) = \mathsf{CCP}(\mathcal{R}) \cup \mathsf{iCCP}(\mathcal{R}) \cup \mathsf{CVP}(\mathcal{R}) \tag{30}$$

which extends $\mathsf{CCP}(\mathcal{R})$ *with the improper conditional critical pairs and conditional variable pairs is the set of* extended conditional critical pairs *of $\mathcal{R}$.*

The following result provides a general characterization of local confluence of conditional rewriting on the basis of the analysis of extended conditional pairs.

Theorem 11. *A CTRS $\mathcal{R}$ is locally confluent iff all pairs in* $\mathsf{ECCP}(\mathcal{R})$ *are joinable.*

Example 12. *For $\mathcal{R}$ in Example 1, consider the conditional variable pair (4):*

$$\langle \mathsf{f}(x'), \mathsf{c}\rangle \Leftarrow x \to x', x \approx \mathsf{a}$$

The sequence $x \to x', x \approx \mathsf{a}$ is clearly feasible *(with $x \mapsto \mathsf{a}$ and $x' \mapsto \mathsf{b}$, for instance). However, the sequence $x \to x', x \approx \mathsf{a}, \mathsf{f}(x') \to^* z, \mathsf{c} \to^* z$ is proved $\overline{\mathcal{R}}$*-infeasible *by* infChecker*. Thus, (4)* is not *joinable. By Theorem 11, $\mathcal{R}$ is not (locally) confluent.*

As a corollary of Theorem 11 and Newman's Lemma, we have the following.

Theorem 13. *A terminating CTRS $\mathcal{R}$ is confluent iff all* pairs *in* $\mathsf{ECCP}(\mathcal{R})$ *are joinable.*

Example 14. *For $\mathcal{R}$ in Example 3, improper CCPs are harmless, and the only proper CCP (26) and CVP (29) are joinable. By Theorem 11, $\mathcal{R}$ is locally confluent. Since $\mathcal{R}$ is terminating (use* MU-TERM *http://zenon.dsic.upv.es/muterm/) by Theorem 13, $\mathcal{R}$ is confluent.*

The CTRSs in Examples 1 and 3 cannot be handled by any automatic tool available in the CoCoWeb platform http://cl-informatik.uibk.ac.at/software/cocoweb/. We are currently implementing the results of this paper as part of the confluence tool CONFident http://zenon.dsic.upv.es/confident/.

References

[1] Jürgen Avenhaus and Carlos Loría-Sáenz. On conditional rewrite systems with extra variables and deterministic logic programs. In *Proc. of LPAR'94*, LNAI 822:215–229, 1994.

[2] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[3] Nachum Dershowitz and Mitsuhiro Okada. A Rationale for Conditional Equational Programming. *Theor. Comput. Sci.*, 75(1&2):111–138, 1990.

[4] Nachum Dershowitz, Mitsuhiro Okada, and G. Sivakumar. Confluence of Conditional Rewrite Systems. In *Proc. of the 1st International Workshop on Conditional Term Rewriting Systems*, LNCS 30831–44, 1987.

[5] Raúl Gutiérrez, Salvador Lucas, and Miguel Vítores. Confluence of Conditional Rewriting in Logic Form. In *Proc. of FSTTCS 2021* LIPIcs 213:44:1–44:18, 2021.

[6] Gérard P. Huet. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems: Abstract Properties and Applications to Term Rewriting Systems. *J. ACM*, 27(4):797–821, 1980.

[7] Stéphane Kaplan. Simplifying conditional term rewriting systems: Unification, termination and confluence. *J. Symb. Comput.*, 4(3):295–334, 1987.

[8] Salvador Lucas. Proving semantic properties as first-order satisfiability. *Artif. Intell.*, 277, 2019.

[9] Enno Ohlebusch. *Advanced Topics in Term Rewriting.* Springer, 2002.

[10] Christian Sternagel and Thomas Sternagel. Certifying confluence of quasi-decreasing strongly deterministic conditional term rewrite systems. In *Proc. of CADE-26*, LNAI 10395:413–43, 2017.

A Critical Pair Criterion for Level-Commutation of Conditional Term Rewriting Systems*

Ryota Haga, Yuki Kagaya, and Takahito Aoto

Niigata University, Niigata, Japan
{ r-haga@nue.,kagaya@nue.,aoto@ }ie.niigata-u.ac.jp

Abstract

We introduce level-commutation of conditional term rewriting systems (CTRSs) that extends the notion of level-confluence, in a way similar to extending confluence to commutation. We show a criterion for level-commutation of oriented CTRSs, which generalizes the one for commutation of term rewriting systems in (Toyama, 1987). As a corollary, we obtain a criterion of level-confluence of oriented CTRSs which extends the one in (Suzuki et al., 1995).

1 Introduction

Level-confluence is a property of conditional term rewriting systems (CTRSs) that implies confluence. Suzuki et al. showed that orthogonal properly oriented right-stable (oriented) CTRSs are level-confluent [6]. In unconditional case, some criteria for left-linear (possibly overlapping) term rewriting systems (TRSs) to have confluence are known (e.g., [3, 8]). But similar extensions for left-linear (possibly overlapping) CTRSs are not known. Also, in unconditional case, several criteria for ensuring commutation for left-linear TRSs $\mathcal{R}$ and $\mathcal{S}$ are known (e.g., [8, 9])—commutation coincides with confluence when $\mathcal{R} = \mathcal{S}$. Again, similar criteria for left-linear CTRSs are not known.

In this paper, we give a critical pair criterion for left-linear properly oriented right-stable CTRSs, under which we prove level-commutation of CTRSs. Our critical pair criterion is a generalization of the one given for TRSs in [8]. As a corollary, we obtain a critical pair criterion for level-confluence of (possibly) overlapping oriented CTRSs, which properly extends the result of [6] mentioned above.

2 Preliminaries

We basically follow standard notions and notations (e.g., [1, 5]). Below, we explain some key notions and fix some notations that will be used in this paper.

The set of variables in a term t is denoted by $\mathcal{V}(t)$. A term t is *linear* if each variable occurs in t at most once; t is *ground* if no variable occurs in t. The set of positions in a term t is denoted by $\mathrm{Pos}(t)$; the *root* position is written as ϵ. The symbol (function symbol or variable) at a position $p \in \mathrm{Pos}(t)$ in a term t is written as $t(p)$. The subterm of t at a position $p \in \mathrm{Pos}(t)$ is written as $t|_p$. We write $t[u]_p$ the term obtained from t by replacing the subterm at the position $p \in \mathrm{Pos}(t)$ with a term u.

If $t = C[u]$ for a context C, we say u is a *subterm* of t. We will speak of subterm occurrences when we consider subterms with their respective positions; see e.g. [7] for a precise formalization of subterm occurrences. We will use capital letters $A, B, \dots$ for subterm occurrences. For

*This work was partly supported by JSPS KAKENHI Grant Number JP21K11750.

simplicity, a subterm occurrence A in a term is also treated as a term A (for example, we might write $A \to_{\mathcal{R}} B$). By a notation $A \subseteq B$, we indicate that A is a subterm occurrence in a subterm occurrence B; thus, for example, $|\{A \mid A \subseteq f(x,x)\}| = 3$.

Each rewrite rule $l \to r$ satisfies the usual variable conditions $l \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. Rewrite rules are identified modulo renaming. A TRS $\mathcal{R}$ is *left-linear* if l is linear for each $l \to r \in \mathcal{R}$. We write $s \to^{p}_{\mathcal{R}} t$ if $s|_p$ is the redex of this rewrite step. We have a *parallel rewrite step* $s \mathrel{+\!\!\!\!\!\!\mapsto}_{\mathcal{R}} t$ if $s = C[s_1,\dots,s_n]$, $t = C[t_1,\dots,t_n]$ ($n \geq 0$) for some contexts C and subterms s_i, t_i such that $s_i \to_{\mathcal{R}} t_i$ for all $i = 1,\dots,n$.

A relation $\to$ is *confluent* if $\overset{*}{\leftarrow} \circ \overset{*}{\to} \subseteq \overset{*}{\to} \circ \overset{*}{\leftarrow}$; A TRS $\mathcal{R}$ is confluent if so is its rewrite relation $\to_{\mathcal{R}}$. Relations $\to$ and $\leadsto$ *commute* (or, are *commutative*) if $\overset{*}{\leftarrow} \circ \overset{*}{\leadsto} \subseteq \overset{*}{\leadsto} \circ \overset{*}{\leftarrow}$; TRSs $\mathcal{R}$ and $\mathcal{S}$ commute if so are their rewrite relations $\to_{\mathcal{R}}$ and $\to_{\mathcal{S}}$. Clearly, self-commutativity equals confluence, and a sufficient criteria for commutativity naturally arises the one for confluence.

Let $l_1 \to r_1$ and $l_2 \to r_2$ be rewrite rules so that their sets of variables are renamed to be disjoint. If a non-variable subterm $l_2|_p$ of l_2 satisfies $l_2|_p\sigma = l_1\sigma$ for some substitution σ, we say that $l_1 \to r_1$ *overlaps on* $l_2 \to r_2$ (at p), provided that $p \neq \epsilon$ for the case $l_1 \to r_1$ and $l_2 \to r_2$ are identical. Suppose $l_1 \to r_1$ overlaps on $l_2 \to r_2$ at p and σ is a mgu of $l_2|_p$ and l_1. Then the pair $\langle l_2[r_1]_p\sigma, r_2\sigma\rangle$ is called a *critical pair* (obtained from that overlap); when $p = \epsilon$, the pair is called *outer* and $p > \epsilon$, the pair is called *inner*. The set of critical pairs from overlaps of rules of $\mathcal{R}$ is denoted by $CP(\mathcal{R})$; the set of outer (inner) critical pairs are denoted by $CP_{out}(\mathcal{R})$ (resp. $CP_{in}(\mathcal{R})$). Let $\mathcal{R}, \mathcal{S}$ be TRSs. The set of critical pairs obtained from overlaps of $l_1 \to r_1 \in \mathcal{R}$ on $l_2 \to r_2 \in \mathcal{S}$ is denoted by $CP(\mathcal{R},\mathcal{S})$. The sets $CP_{out}(\mathcal{R},\mathcal{S})$ and $CP_{in}(\mathcal{R},\mathcal{S})$ are defined similarly. We are now ready to state a sufficient criteria for commutativity of TRSs.

Proposition 1 ([8]). *Let $\mathcal{R}$ and $\mathcal{S}$ be left-linear TRSs. If both of the following conditions are satisfied, then $\mathcal{R}$ and $\mathcal{S}$ commute:*

1. *for any $\langle p,q\rangle \in CP(\mathcal{R},\mathcal{S})$, there exists s such that $p \mathrel{+\!\!\!\!\!\!\mapsto}_{\mathcal{S}} s$ and $q \overset{*}{\to}_{\mathcal{R}} s$, and*
2. *for any $\langle q,p\rangle \in CP_{in}(\mathcal{S},\mathcal{R})$, $q \mathrel{+\!\!\!\!\!\!\mapsto}_{\mathcal{R}} p$ holds.*

We note that the above criterion for commutativity arises a criterion for confluence: a left-linear TRS $\mathcal{R}$ if confluent if (1) for any $\langle p,q\rangle \in CP_{out}(\mathcal{R})$, there exists s such that $p \mathrel{+\!\!\!\!\!\!\mapsto}_{\mathcal{R}} s$ and $q \overset{*}{\to}_{\mathcal{R}} s$, and (2) for any $\langle q,p\rangle \in CP_{in}(\mathcal{R})$, $q \mathrel{+\!\!\!\!\!\!\mapsto}_{\mathcal{R}} p$ holds. Note here in the condition (1), considering $\langle p,q\rangle \in CP_{out}(\mathcal{R})$ is sufficient, instead of considering $\langle p,q\rangle \in CP(\mathcal{R})$, because of the presence of condition (2).

A *conditional rewrite rule* has the form $l \to r \Leftarrow u_1 \approx v_1,\dots,u_k \approx v_k$ where $l \notin \mathcal{V}$; here, $u_1 \approx v_1,\dots,u_k \approx v_k$ is a sequence of (directed) equations, called the conditional part of the rule. Let $c = u_1 \approx v_1,\dots,u_k \approx v_k$. Then, for any given substitution σ, we write $c\sigma = u_1\sigma \approx v_1\sigma,\dots,u_k\sigma \approx v_k\sigma$. We often also treat c as a set $\{u_1 \approx v_1,\dots,u_k \approx v_k\}$ so as to write $u \approx v \in c$, $c\sigma \subseteq \leadsto$, etc, whose meaning should be apparent. The empty sequence is also written as $\emptyset$, and $l \to r \Leftarrow \emptyset$ is abbreviated as $l \to r$. Conditional rewrite rules are also identified modulo renaming.

A rewrite step of *oriented* CTRS $\mathcal{R}$ is defined via TRS $\mathcal{R}_i$ ($i \in \omega$), which are inductively given as follows:

$$\begin{array}{lcl} \mathcal{R}_0 & = & \emptyset \\ \mathcal{R}_{n+1} & = & \{l\sigma \to r\sigma \mid l \to r \Leftarrow c \in \mathcal{R}, c\sigma \subseteq \overset{*}{\to}_{\mathcal{R}_n}\} \end{array}$$

A rewrite step $s \to_{\mathcal{R}} t$ of CTRS $\mathcal{R}$ is given as $s \to_{\mathcal{R}} t$ iff $s \to_{\mathcal{R}_n} t$ for some n. The smallest n such that $s \to_{\mathcal{R}_n} t$ is called the *level* of the rewrite step $s \to_{\mathcal{R}} t$. We also use the notation $\to_{\mathcal{R}_{<n}} = \bigcup_{i<n} \to_{\mathcal{R}_i}$. *We will only consider oriented CTRSs in this paper, and so we will abbreviate oriented CTRSs as CTRSs henceforth.*

A CTRS $\mathcal{R}$ is *level-confluent* if TRSs $\mathcal{R}_n$ are confluent for all $n \geq 0$. Clearly, level-confluence implies confluence. One can naturally extend the notion of level-confluence, in the similar way extending confluence to commutation.

Definition 2. *CTRSs $\mathcal{R}$ and $\mathcal{S}$ are* level-commutative *if for any $m, n \geq 0$, $\xleftarrow{*}_{\mathcal{R}_m} \circ \xrightarrow{*}_{\mathcal{S}_n} \subseteq \xrightarrow{*}_{\mathcal{S}_n} \circ \xleftarrow{*}_{\mathcal{R}_m}$.*

Clearly, level-commutativity implies commutativity, and self-level-commutativity equals level-confluence.

We refer some notions, such as properly orientedness, right-stability and 3-CTRSs, necessary to give a sufficient criteria for level-confluence to [6]. A CTRS $\mathcal{R}$ is left-linear if l is linear for all $l \to r \Leftarrow c \in \mathcal{R}$. Let $l_1 \to r_1 \Leftarrow c_1$ and $l_2 \to r_2 \Leftarrow c_2$ be conditional rewrite rules so that their sets of variables are renamed to be disjoint. We say $l_1 \to r_1 \Leftarrow c_1$ overlaps on $l_2 \to r_2 \Leftarrow c_2$ (at p) if non-variable subterm $l_2|_p$ of l_2 satisfies $l_2|_p\sigma = l_1\sigma$ for some substitution σ, provided that $p \neq \epsilon$ for the case $l_1 \to r_1 \Leftarrow c_1$ and $l_2 \to r_2 \Leftarrow c_2$ are identical. A CTRS $\mathcal{R}$ is *non-overlapping* if there is no overlap between rules of $\mathcal{R}$; $\mathcal{R}$ is *orthogonal* if it is left-linear and non-overlapping.

Proposition 3 ([6])**.** *Let $\mathcal{R}$ be an orthogonal, properly oriented, right-stable 3-CTRS. Then, $\xleftarrow{*}_{\mathcal{R}_m} \circ \xrightarrow{*}_{\mathcal{R}_n} \subseteq \xrightarrow{*}_{\mathcal{R}_n} \circ \xleftarrow{*}_{\mathcal{R}_m}$ for any $m, n \geq 0$. In particular, $\mathcal{R}$ is level-confluent.*

Proposition 1 only deals with TRSs but capable of non-orthogonal case. On the other hand, Proposition 3 can deal with CTRSs (not only TRSs) but limited to only orthogonal case. Also Proposition 3 only claims on (level-)confluence, whereas Proposition 1 claims commutativity. We will show how to unify these two propositions in the next section.

3 Level-Commutation

Let $\mathcal{R}$ be a CTRS. We use the notion of extended parallel rewriting [6] given as follows: we write $s \mathrel{\hookrightarrow\!\!\!\!\!\!\!+\!\!\!+}_{\mathcal{R}_n} t$ if $s = C[A_1, \ldots, A_p]$, $t = C[B_1, \ldots, B_p]$ $(p \geq 0)$ for some context C and subterm occurrences A_i, B_i such that either $A_i \to^{\epsilon}_{\mathcal{R}_n} B_i$ or $A_i \xrightarrow{*}_{\mathcal{R}_{<n}} B_i$ for all $i = 1, \ldots, p$. We put $\mathrel{\hookrightarrow\!\!\!\!\!\!\!+\!\!\!+}_{\mathcal{R}} = \bigcup_{n \geq 0} \mathrel{\hookrightarrow\!\!\!\!\!\!\!+\!\!\!+}_{\mathcal{R}_n}$, which is called the *extended parallel rewrite step* of $\mathcal{R}$. We will also write $s \overset{A_1,\ldots,A_p}{\mathrel{\hookrightarrow\!\!\!\!\!\!\!+\!\!\!+}}_{\mathcal{R}} t$ to indicate subterm occurrences $A_1, \ldots, A_p$.

Proposition 3 is obtained in [6] by showing that if $t \overset{*}{\mathrel{\hookleftarrow\!\!\!\!\!\!\!+\!\!\!+}}_{\mathcal{R}_m} t_1$ and $t \overset{*}{\mathrel{\hookrightarrow\!\!\!\!\!\!\!+\!\!\!+}}_{\mathcal{R}_n} t_2$ then there exists t_3 such that $t_1 \overset{*}{\mathrel{\hookrightarrow\!\!\!\!\!\!\!+\!\!\!+}}_{\mathcal{R}_n} t_3$ and $t_2 \overset{*}{\mathrel{\hookrightarrow\!\!\!\!\!\!\!+\!\!\!+}}_{\mathcal{R}_m} t_3$. Our first key ingredient is our proof scenario showing that if $t \overset{*}{\mathrel{\hookleftarrow\!\!\!\!\!\!\!+\!\!\!+}}_{\mathcal{R}_m} t_1$ and $t \overset{*}{\mathrel{\hookrightarrow\!\!\!\!\!\!\!+\!\!\!+}}_{\mathcal{S}_n} t_2$ then there exists t_3 such that $t_1 \mathrel{\hookrightarrow\!\!\!\!\!\!\!+\!\!\!+}_{\mathcal{S}_n} \circ \overset{*}{\mathrel{\hookrightarrow\!\!\!\!\!\!\!+\!\!\!+}}_{\mathcal{S}_{<n}} t_3$ and $t_2 \xrightarrow{*}_{\mathcal{R}_m} t_3$. We now reason why this approach is sound using an abstract setting.

Let $(\to_n)_{n \in \mathbb{N}}$ be ($\mathbb{N}$-indexed) relations on a set X. We put $\to_{<n} = \bigcup_{i<n} \to_i$. The $\mathbb{N}$-indexed relations $(\to_n)_{n \in \mathbb{N}}$ are said to be *up-simulated* if $\xrightarrow{*}_{<n} \subseteq \to_n$ for any $n \in \mathbb{N}$.

Lemma 4. *Let $(\to_n)_{n \in \mathbb{N}}, (\rightsquigarrow_n)_{n \in \mathbb{N}}$ be up-simulated relations on a set X. Suppose that, for any $m, n \in \mathbb{N}$, $\leftarrow_m \circ \rightsquigarrow_n \subseteq \rightsquigarrow_n \circ \overset{*}{\rightsquigarrow}_{<n} \circ \xleftarrow{*}_m$. Then $\xleftarrow{*}_m \circ \rightsquigarrow_n \subseteq \overset{*}{\rightsquigarrow}_n \circ \xleftarrow{*}_m$ for any $m, n \in \mathbb{N}$.*

Proof. Suppose $z \xleftarrow{*}_m x \rightsquigarrow_n y$ and let $k = |x \xrightarrow{*}_m z|$. Then use by induction on $\langle n + m, k \rangle$ to show $z \rightsquigarrow_n \circ \overset{*}{\rightsquigarrow}_{<n} w \xleftarrow{*}_m y$. ☐

Lemma 5. *Let $(\to_n)_{n \in \mathbb{N}}, (\rightsquigarrow_n)_{n \in \mathbb{N}}$ be up-simulated relations on a set X. Suppose that, for any $m, n \in \mathbb{N}$, $\leftarrow_m \circ \rightsquigarrow_n \subseteq \rightsquigarrow_n \circ \overset{*}{\rightsquigarrow}_{<n} \circ \xleftarrow{*}_m$. Then $\xleftarrow{*}_m \circ \overset{*}{\rightsquigarrow}_n \subseteq \overset{*}{\rightsquigarrow}_n \circ \xleftarrow{*}_m$ for any $m, n \in \mathbb{N}$.*

Proof. Use induction on $|\overset{*}{\rightsquigarrow}_n|$ and Lemma 4. ☐

From this lemma, it easily follows:

Lemma 6. *Let $\mathcal{R}, \mathcal{S}$ be CTRSs. Suppose $\mathrel{\leftarrow\!\!\shortmid\!\shortmid\!\!\hookleftarrow}_{\mathcal{R}_m} \circ \mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}_{\mathcal{S}_n} \subseteq \mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}_{\mathcal{S}_n} \circ \overset{*}{\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}}_{\mathcal{S}_{<n}} \circ \overset{*}{\leftarrow}_{\mathcal{R}_m}$ for any $m, n \geq 0$. Then, for any m, n, we have $\overset{*}{\leftarrow}_{\mathcal{R}_m} \circ \overset{*}{\rightarrow}_{\mathcal{S}_n} \subseteq \overset{*}{\rightarrow}_{\mathcal{S}_n} \circ \overset{*}{\leftarrow}_{\mathcal{R}_m}$.*

Proof. We now abbreviate $\mathcal{R}$ and $\mathcal{S}$ for readability. Suppose $z \overset{*}{\leftarrow}_m x \overset{*}{\rightarrow}_n y$. As $\rightarrow_k \subseteq \mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}_k$ for each k, we have $z \overset{*}{\mathrel{\leftarrow\!\!\shortmid\!\shortmid\!\!\hookleftarrow}}_m x \overset{*}{\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}}_n y$. Clearly, $(\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}_k)_{k\in\mathbb{N}}$ are up-simulated by its definition. Thus, it follows $z \overset{*}{\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}}_n v \overset{*}{\mathrel{\leftarrow\!\!\shortmid\!\shortmid\!\!\hookleftarrow}}_m y$ by using Lemma 5 and our hypothesis. Because $\overset{*}{\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}}_k \subseteq \overset{*}{\rightarrow}_k$ for each k, we obtain $z \overset{*}{\rightarrow}_n v \overset{*}{\leftarrow}_m y$. $\square$

Our second key ingredient is the following alternative definition of CCP.

Definition 7 (condition-separated CCP)**.** *Suppose $l_1 \to r_1 \Leftarrow c_1$ overlaps on $l_2 \to r_2 \Leftarrow c_2$ at p and σ is a mgu of $l_2|_p$ and l_1. Then the quadruple $\langle l_2[r_1]_p\sigma, r_2\sigma\rangle \Leftarrow \langle c_1\sigma, c_2\sigma\rangle$ is called a* (condition-separated) conditional critical pair (CCP, for short) *(obtained from that overlap); when $p = \epsilon$, the pair is called* outer *and $p > \epsilon$, the pair is called* inner*. The set of (outer, inner) critical pairs obtained from overlaps of $l_1 \to r_1 \Leftarrow c_1 \in \mathcal{R}$ on $l_2 \to r_2 \Leftarrow c_2 \in \mathcal{S}$ is denoted by $CCP(\mathcal{R}, \mathcal{S})$ (resp. $CCP_{out}(\mathcal{R}, \mathcal{S})$, $CCP_{in}(\mathcal{R}, \mathcal{S})$). The set of (outer, inner) critical pairs from overlaps of rules of $\mathcal{R}$ is denoted by $CCP(\mathcal{R})$ (resp. $CCP_{out}(\mathcal{R})$, $CCP_{in}(\mathcal{R})$).*

We note that, in literature, instead of distinguishing two sequences $c_1\sigma$ and $c_2\sigma$, the combined sequence of $c_1\sigma$ and $c_2\sigma$ is employed in the definition of CCPs.

Now we present our critical pair criterion for commutativity.

Theorem 8. *Let $\mathcal{R}$ and $\mathcal{S}$ be left-linear, properly oriented, right-stable 3-CTRSs. If the following conditions are satisfied, then $\mathcal{R}$ and $\mathcal{S}$ are level-commutative:*

1. *for any $\langle u, v\rangle \Leftarrow \langle c, c'\rangle \in CCP(\mathcal{R}, \mathcal{S})$, $m, n \geq 1$ and substitution ρ, if $c\rho \subseteq \overset{*}{\rightarrow}_{\mathcal{R}_{m-1}}$ and $c'\rho \subseteq \overset{*}{\rightarrow}_{\mathcal{S}_{n-1}}$ then there exists s such that $u\rho \mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}_{\mathcal{S}_n} \circ \overset{*}{\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}}_{\mathcal{S}_{<n}} s$ and $v\rho \overset{*}{\rightarrow}_{\mathcal{R}_m} s$, and*

2. *for any $\langle v, u\rangle \Leftarrow \langle c', c\rangle \in CCP_{in}(\mathcal{S}, \mathcal{R})$, $m, n \geq 1$ and substitution ρ, if $c\rho \subseteq \overset{*}{\rightarrow}_{\mathcal{R}_{m-1}}$ and $c'\rho \subseteq \overset{*}{\rightarrow}_{\mathcal{S}_{n-1}}$ then $v\rho \mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}_{\mathcal{R}_m} u\rho$.*

The detailed proof (in Japanese) can be found in [2], which extends [4]. We here only give a brief sketch of the proof.

Proof. Let $M \overset{A_1,\ldots,A_m}{\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}}_{\mathcal{R}_m} N$ and $M \overset{B_1,\ldots,B_n}{\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}}_{\mathcal{S}_n} P$. We show $N \mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}_{\mathcal{S}_n} \circ \overset{*}{\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}}_{<\mathcal{S}_n} Q$ and $P \overset{*}{\rightarrow}_{\mathcal{R}_m} Q$ for some Q. Let $\Gamma = \{A_i \mid \exists B_j.\ A_i \subseteq B_j\} \cup \{B_i \mid \exists A_j.\ B_i \subseteq A_j\}$ and $\Delta = \{A_i \mid \forall B_j.\ A_i \not\subseteq B_j\} \cup \{B_i \mid \forall A_j.\ B_i \not\subseteq A_j\}$. Let $|\Gamma| = \sum_{C\in\Gamma} |C|$. Thus, $|\Gamma|$ is the sum of size of overlaps and Δ is the set of maximal redexes. Our proof proceeds on induction on lexicographic combination of $\langle m+n, |\Gamma|\rangle$. The cases for $m = 0$ or $n = 0$ are easy, thus we consider the cases for $m > 0, n > 0$. Let $\Delta = \{M_1, \ldots, M_p\}$. Then we have $M = C[M_1, \ldots, M_p]$ for some context C. Furthermore, we have $N = C[N_1, \ldots, N_p]$ and $P = C[P_1, \ldots, P_p]$ for some $N_1, \ldots, N_p, P_1, \ldots, P_p$ such that $M_i \mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}_{\mathcal{R}_m} N_i$, $M_i \mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}_{\mathcal{S}_n} P_i$ $(i = 1, \ldots, p)$. Thus, it suffices to show for each M_i, there exists Q_i such that $N_i \mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}_{\mathcal{S}_n} \circ \overset{*}{\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}}_{<\mathcal{S}_n} Q_i$ and $P_i \overset{*}{\rightarrow}_{\mathcal{R}_m} Q_i$. We distinguish two cases:

1. Case $M_i \notin \{B_1, \ldots, B_n\}$. Let $\{B'_1, \ldots, B'_q\} = \{B_j \mid 1 \leq j \leq n, B'_j \subset M_i\}$. Then we have $M_i = C_i[B'_1, \ldots, B'_q]$ and $P_i = C_i[\tilde{B}'_1, \ldots, \tilde{B}'_q]$ so that $M_i \overset{M_i}{\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}}_{\mathcal{R}_m} N_i$ and $M_i \overset{B'_1,\ldots,B'_q}{\mathrel{\hookrightarrow\!\!\shortmid\!\shortmid\!\!\rightarrow}}_{\mathcal{S}_n} P_i$. We distinguish the cases.

(a) Case $M_i \xrightarrow{*}_{\mathcal{R}_{m-1}} N_i$. This case basically follows from the induction hypothesis.

(b) Case $M_i \xrightarrow{M_i}_{\mathcal{R}_m} N_i$. Then we have $M_i = l\theta$, $N_i = r\theta$ and $\mathcal{R}_{m-1} \vdash c\theta$ for some $l \to r \Leftarrow c \in \mathcal{R}$. If all redex occurrences B'_j in M_i is contained in the substitution θ, then the desired Q_i exists. Suppose otherwise, i.e. there exists B'_j which is not contained in θ. Let X and Y be sets given by

$$\begin{array}{rcl} X & = & \{B'_j \mid 1 \leq j \leq q, B'_j \text{ is not contained in } \theta\} \\ Y & = & \{B'_j \mid 1 \leq j \leq q, B'_j \text{ is contained in } \theta\} \end{array}$$

For each $B'_j \in X$, either $B'_j \xrightarrow{B'_j}_{\mathcal{S}_n} \tilde{B}'_j$ or $B'_j \xrightarrow{*}_{<\mathcal{S}_n} \tilde{B}'_j$. We distinguish two cases.

i. Case that there exists $B'_j \in X$ such that $B'_j \xrightarrow{B'_j}_{\mathcal{S}_n} \tilde{M}_i (= \tilde{B}'_j)$. This case follows using the condition 2 of the theorem.

ii. Case that $B'_j \xrightarrow{*}_{\mathcal{S}_{n-1}} \tilde{M}_i (= \tilde{B}'_j)$ holds for any $B'_j \in X$. As $M_i \mathrel{\overset{B'_1,\ldots,B'_q}{\hookrightarrow\!\!\!\!\!\!\!\shortparallel\;}}_{\mathcal{S}_n} P_i$ and $B'_1, \ldots, B'_q$ are parallel, we can first rewrite $B'_j \in Y (1 \leq j \leq q)$. That is, let $Y = \{B''_1, \ldots, B''_r\}$, and we have $M_i \mathrel{\overset{B''_1,\ldots,B''_r}{\hookrightarrow\!\!\!\!\!\!\!\shortparallel\;}}_{\mathcal{S}_n} \hat{M}_i \xrightarrow{*}_{\mathcal{S}_{n-1}} P_i$. Here, since each B''_j in contained in the substitution θ, one can obtain $\hat{Q}$ such that $N_i \mathrel{\hookrightarrow\!\!\!\!\!\!\!\shortparallel\;}_{\mathcal{S}_n} \hat{Q}$, $\hat{M}_i \to_{\mathcal{R}_m} \hat{Q}$. Furthermore, since $\to_{\mathcal{R}_m} \subseteq \mathrel{\hookrightarrow\!\!\!\!\!\!\!\shortparallel\;}_{\mathcal{R}_m}$ and $\xrightarrow{*}_{\mathcal{S}_{n-1}} \subseteq \mathrel{\overset{*}{\hookrightarrow\!\!\!\!\!\!\!\shortparallel\;}}_{\mathcal{S}_{n-1}}$, using induction hypothesis, we can obtain Q_i such that $\hat{Q} \leftarrow_{\mathcal{R}_m} \hat{M}_i \xrightarrow{*}_{\mathcal{S}_{n-1}} P_i$ and $\hat{Q} \mathrel{\overset{*}{\hookrightarrow\!\!\!\!\!\!\!\shortparallel\;}}_{\mathcal{S}_{n-1}} Q_i$, $P_i \xrightarrow{*}_{\mathcal{R}_m} Q_i$. This is the case where our first key ingredient becomes necessary.

2. Case $M_i \in \{B_1, \ldots, B_n\}$. Let $\{A'_1, \ldots, A'_q\} = \{A_j \mid 1 \leq j \leq n, A'_j \subseteq M_i\}$. Then one can put $M_i = C_i[A'_1, \ldots, A'_q]$, $N_i = C_i[\tilde{A}'_1, \ldots, \tilde{A}'_q]$, $M_i \mathrel{\overset{A'_1,\ldots,A'_q}{\hookrightarrow\!\!\!\!\!\!\!\shortparallel\;}}_{\mathcal{R}_m} N_i$ and $M_i \mathrel{\overset{M_i}{\hookrightarrow\!\!\!\!\!\!\!\shortparallel\;}}_{\mathcal{S}_n} P_i$. By definition, $M_i \mathrel{\overset{M_i}{\hookrightarrow\!\!\!\!\!\!\!\shortparallel\;}}_{\mathcal{S}_n} P_i$ is either of the form $M_i \xrightarrow{*}_{\mathcal{S}_{n-1}} P_i$ or $M_i \xrightarrow{M_i}_{\mathcal{S}_n} P_i$. In the latter case, there exists $l' \to r' \Leftarrow c' \in \mathcal{S}$ such that $M_i = l'\theta'$, $N_i = r'\theta'$ and $c'\theta' \subseteq \xrightarrow{*}_{\mathcal{S}_{n-1}}$.
We distinguish whether all redex occurrences A'_j in M_i is contained in θ' or not. In the latter case, w.l.o.g. assume that A'_1 is not contained in θ'. Then there exists $l \to r \Leftarrow c \in \mathcal{R}$ such that $A'_1 = l\theta$ and $c\theta \subseteq \xrightarrow{*}_{\mathcal{R}_{m-1}}$. We further distinguish two cases:

(α) $A'_1 = M_i$ and $l \to r \Leftarrow c \in \mathcal{R}$ are $l' \to r' \Leftarrow c' \in \mathcal{S}$ are identical.

(β) $A'_1 \neq M_i$ or $l \to r \Leftarrow c \in \mathcal{R}$ and $l' \to r' \Leftarrow c' \in \mathcal{S}$ are distinct.

Case of (α), we use a construction similar to [6]. Here, we also use our assumption that $\mathcal{R}$ and $\mathcal{S}$ are properly oriented and right-stable. Case of (β), the condition 1 of the theorem is used.

Finally, from Lemma 6 we conclude that $\mathcal{R}$ and $\mathcal{S}$ are level-commutative. □

A level-confluence criterion is obtain by taking $\mathcal{R} = \mathcal{S}$:

Corollary 9. *Let $\mathcal{R}$ be a left-linear, properly oriented, right-stable 3-CTRSs. If the following conditions are satisfied, then $\mathcal{R}$ is level-confluent:*

1. *for any $\langle u, v\rangle \Leftarrow \langle c, c'\rangle \in CCP_{out}(\mathcal{R})$, $m, n \geq 1$ and substitution ρ, if $c\rho \subseteq \xrightarrow{*}_{\mathcal{R}_{m-1}}$ and $c'\rho \subseteq \xrightarrow{*}_{\mathcal{R}_{n-1}}$ then there exists s such that $u\rho \mathrel{\hookrightarrow\!\!\!\!\!\!\!\shortparallel\;}_{\mathcal{S}_n} \circ \mathrel{\overset{*}{\hookrightarrow\!\!\!\!\!\!\!\shortparallel\;}}_{\mathcal{R}_{<n}} s$ and $v\rho \xrightarrow{*}_{\mathcal{R}_m} s$, and*

2. *for any* $\langle v, u\rangle \Leftarrow \langle c', c\rangle \in CCP_{in}(\mathcal{R})$, $m, n \geq 1$ *and substitution* ρ, *if* $c\rho \subseteq \xrightarrow{*}_{\mathcal{R}_{m-1}}$ *and* $c'\rho \subseteq \xrightarrow{*}_{\mathcal{R}_{n-1}}$ *then* $v\rho \hookrightarrow\!\!\!\!\!\twoheadrightarrow_{\mathcal{R}_m} u\rho$.

Example 10. *Let* $\mathcal{R}$ *and* $\mathcal{S}$ *be the following CTRSs:*

$$\mathcal{R} = \left\{ \begin{array}{lcl} p(x) \to q(x) & & \\ r(x) \to s(p(x)) & & \\ s(x) \to f(y) & \Leftarrow & p(x) \approx y \end{array} \right\} \quad \mathcal{S} = \left\{ \begin{array}{lcl} p(x) \to r(x) & & \\ q(x) \to s(p(x)) & & \\ s(x) \to f(y) & \Leftarrow & p(x) \approx y \end{array} \right\}$$

We have $CCP(\mathcal{R}, \mathcal{S}) = \{\langle q(x), r(x)\rangle \Leftarrow \langle \emptyset, \emptyset\rangle\}$ *and* $CCP_{in}(\mathcal{S}, \mathcal{R}) = \emptyset$. *Note that the overlap of* $s(x) \to f(y) \Leftarrow p(x) \approx y \in \mathcal{R}$ *and* $s(x) \to f(y) \Leftarrow p(x) \approx y \in \mathcal{S}$ *is not considered, as these rules are identical; the case 2* (α) *of the proof sketch above treats this case. Now, because we have* $q(x) \to_{\mathcal{S}} s(p(x))$ *and* $r(x) \to_{\mathcal{R}} s(p(x))$, *the condition (1) of the Theorem 8 is satisfied. Other conditions of the theorem are also satisfied. Thus,* $\mathcal{R}$ *and* $\mathcal{S}$ *are level-commutative. Similarly, one can show* $\mathcal{R} \cup \mathcal{S}$ *is level-confluent.*

Since TRSs can be regarded as CTRSs with no conditions and they are trivially properly-oriented, right-stable, and of type 3, this theorem covers Proposition 1. Since rewrite steps of TRSs are level 1 rewrite steps in CTRSs, however, when restricting to TRSs, Theorem 8 reduces to Proposition 1. On the other hand, Corollary 9 properly extends Proposition 3, as witnessed by $\mathcal{R} \cup \mathcal{S}$ in Example 10.

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[2] R. Haga and T. Aoto. Level-commutativity of conditonal term rewriting systems based on a critical pair criterion. In *Proc. of the 24th JSSST Workshop on Programming and Programming Languages (PPL 2022)*, 2022. In Japanese.

[3] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[4] Y. Kagaya and T. Aoto. Proving level-confluece of conditional term rewriting sytems. In *Proc. of the 22nd JSSST Workshop on Programming and Programming Languages (PPL 2020)*, 2020. In Japanese.

[5] E. Ohlebusch. *Advanced Topins in Term Rewriting.* Springer, 2002.

[6] T. Suzuki, A. Middeldorp, and T. Ida. Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In *Proc. of 6th RTA*, volume 914 of *LNCS*, pages 179–193. Springer-Verlag, 1995.

[7] Terese. *Term Rewriting Systems.* Cambridge University Press, 2003.

[8] Y. Toyama. Commutativity of term rewriting systems. In *Programming of Future Generation Computer II*, pages 393–407, North-Holland, Amsterdam, 1987.

[9] J. Yoshida, T. Aoto, and Y. Toyama. Automating confluence check of term rewriting systems. *Computer Software*, 26(2):76–92, 2009. In Japanese.

Uniform Completeness

Vincent van Oostrom*

Department of Computer Science, University of Bath, United Kingdom
vvo21@bath.ac.uk

Abstract

We introduce uniform completeness and give a local characterisation of it. We show it yields a complete method for showing completeness of rewrite systems.

1 Introduction

The canonical way to establish completeness of rewrite systems is by showing local confluence and termination, as enabled by Newman's Lemma [4, Theorem 3]. In [5] we provided the following alternative route to establishing completeneness:

Corollary 1. *If $\to$ is normalising (WN) and ordered weak Church–Rosser, then $\to$ is complete.*

Proof. Ordered weak Church–Rosser (Definition 3) entails by [5, Theorem 3] that $\to$ has *random descent*,[1] i.e. that any object convertible to normal form reduces to it, and always in the same number of steps. Hence the additional assumption of normalisation entails completeness. □

Observe that this alternative route in fact yields a result stronger than completeness, namely that all reductions from a given object to normal form are not only terminating and end in the same object, but also that the (length) measure of these reductions is always the same.

In this short paper we show the alternative route to be complete, if $\to$ is complete then $\to$ is normalising and ordered weak Church–Rosser, when the order-constraint in the latter is allowed to depend on an arbitrary *measure* [8]. We showcase the technique by some simple examples.

Example 1. *Consider the rewrite system $\to$ having steps $a \to b$, $b \to c$ and $a \to c$. It is trivially complete. Measuring the steps by natural numbers (with addition) as $a \to_1 b$, $b \to_1 c$ and $a \to_2 c$, yields a system that is ordered weak Church–Rosser in the sense of [8] (for this chosen measure). In particular, all reductions from a to its normal form c have the same measure 2. However, $\to$ is not ordered weak Church–Rosser in the sense of [5], i.e. measuring the numbers of steps, since the* length *of the reduction $a \to b \to c$ from a to c is then 2, whereas that of $a \to c$ is 1.*

Note that adjoining a step $c \to c$ in the example yields a system that although no longer WN, still *is* ordered weak Church–Rosser and that for *any* measure, also for the length measure. The following two examples are WN, and ordered weak Church–Rosser for the length measure.

Example 2 ([5, Example 7]). *Consider the rewrite system $\to$ that sorts strings of letters by repeatedly swapping adjacent out-of-order letters. It is easy to see that if $s \leftarrow t \to u$, then $s \to^n t \;{}^n\!\!\leftarrow u$, with $n \in \{0,1,2\}$ depending on whether the respective swaps are the same, non-overlapping, or overlapping. For instance, for $bca \leftarrow cba \to cab$ we have $bca \to bac \to abc \leftarrow acb \leftarrow cab$. Therefore $\to$ is ordered weak-Church–Rosser. As $\to$ is normalising, e.g. by bubble-sort, it terminates uniquely by Corollary 1, taking the same number of steps given a string.*

*Supported by EPSRC Project EP/R029121/1 Typed lambda-calculi with sharing and unsharing.

[1] So named in [5] to honour [4]. The notion was given a *description but not a definition* [9] by Newman.

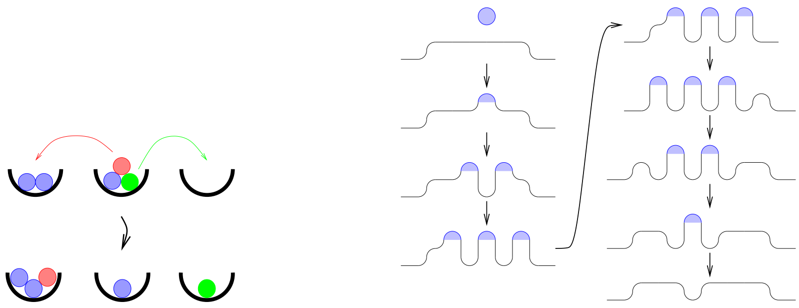


Figure 1: Bowls and beans move (left) and normalisation (right)

Example 3 (cf. [2, Example 8.3]). *Consider the rewrite system $\to$ that given a two-sided infinite sequence $\mathbb{Z} \to \mathbb{N}$ of* bowls *each holding a number of* beans *may, for a bowl holding at least two beans, move one bean to each adjacent bowl, see Figure 1 left. Formally, $t \to s$ if s is the same as t except that for some $z \in \mathbb{Z}$ such that $t(z) \geq 2$, we have $s(z-1) = t(z-1)+1$, $s(z) = t(z)-2$ and $s(z+1) = t(z+1)+1$. It is easy to see that if $s \leftarrow t \to u$, then $s \to^{n} t \;{}^{n}{\leftarrow}\; u$, with $n \in \{0,1\}$ depending on whether the respective moves are the same or not. Therefore $\to$ is ordered weak-Church–Rosser. That $\to$ is normalising follows e.g. by noting that a sequence is in normal form iff each bowl holds at most one bean, and that adding a single bean to a normal form produces 'waves' that first extend outwardly and then, after reaching their limit, die down inwardly, see Figure 1 right. By Corollary 1 $\to$ is complete and by the observation, given a sequence normalising it always takes the same number of steps before reaching the normal form.*

2 Uniformly complete ⇔ has peak random descent

For a property Π of objects of a rewrite system its restriction to *meaningful* objects is of interest, with the crudest approximation of meaningful being that objects be convertible to normal form.

Definition 1. *A rewrite system $\to$ is* uniformly Π *if $\Pi(a)$ for all a with $a \leftrightarrow^* \cdot \not\to$.*

Uniformity resides in there being no steps between the objects that have property Π and those that do not. Obviously, since *uniform Π-ety* requires Π to hold only for the objects convertible to normal form, it is in general weaker than Π. In the literature *uniform termination* has been studied, e.g. in [3] (cf. Remark 1). Here we introduce and study *uniform completeness*, i.e. uniform Π-ety for Π the property defined by $\Pi(a) := a$ *is confluent and terminating.* We relate uniform completeness to extant notions from rewriting [3, 10].

Proposition 1. *$\to$ is uniformly complete iff $\to$ both is uniformly terminating and has NF.*

Proof. For the only–if-direction, suppose $\to$ is uniformly complete. If an object is normalising, it is convertible to some normal form hence terminating by assumption. If an object is convertible to any normal form it is confluent by assumption so reduces to it, i.e. it has NF (the normal form property [10, Definition 1.1,13(iv)]). For the if-direction, suppose $\to$ is uniformly terminating and has NF. Then if an object is convertible to normal form, it is terminating by uniform termination, and the reduction must end in the normal form by NF. ☐

Example 4. *Since β-reduction is confluent it has NF Combined with uniform termination of the λI-calculus [1, p.20 7XXV] and of the simply typed λ-calculus it yields uniform completeness of both. The untyped λ-calculus is not uniformly complete; cf. $(\lambda x.y)\,((\lambda z.z\,z)\,(\lambda u.u\,u))$.*

Remark 1. *For* reduction-closed *properties* Π*, i.e.* if $\Pi(a)$ and $a \to b$ then $\Pi(b)$*, such as termination, a rewrite system being uniformly* Π *is equivalent to the absence of so-called* Π-critical *steps, steps from an object not having property* Π *to one having it. In fact, that characterisation was used as the* definition *of uniform termination in [3, Definition 2].*[2] *Absence of* Π*-critical steps can be positively stated as that all steps are* Π-perpetual*, i.e. preserve* $\neg\Pi$*, cf. [3].*

The key result of this short note is a *local* [4] characterisation of uniform completeness via the notion of peak random descent as introduced in [8, Definition 22]. Peak random descent expresses that for any peak of reductions where the first ends in a normal form, the second can be extended by a reduction to the same normal form such that both resulting legs have the same measure, for a given measure on steps. Here, each step is measured by assigning to it some element of a monoid, distinct from the unit. This is then naturally extended to (finite) reductions by using the unit and operation of the monoid from tail to head, e.g. the measure of $\to_1 \cdot \to_2$ in the monoid of natural numbers with zero and addition is $0+2+1=3$. In order to formalise the notion of peak random descent, we equip rewrite systems with such a measure [8].

Definition 2. *A monoid with addition* $+$ *and zero* $\bot$ *is a* derivation *monoid if it comes equipped with a well-founded partial order* $\leq$ *such that* $\bot$ *is least and* $+$ *is monotonic in both arguments and strictly so in its second. A* measure *for a rewrite system is a map from steps to the non-*$\bot$*-elements of a derivation monoid. The measure of a finite reduction is the sum of the measures of the steps in it, from tail to head. This is extended to infinite reductions by representing these as* steps *of the rewrite system* $\to^\infty$ *[8, Definition 10] having a step from* a *to* b *for* any *infinite* $\to$*-reduction from* a *and* any b*. Such* $\to^\infty$*-steps are measured by* $\top$ *with* $\top$ *added as a top to the monoid.*[3] *That allows to represent reductions that may be either finite or infinite by* $\to^\circledast := (\to \cup \to^\infty)^*$*, called* extended *reduction in [8].*

We use $\mu, \nu, \ldots$ to denote arbitrary measures and $m, n, \ldots$ to denote *finite* ($\neq \top$) ones.

Example 5.

- *The ordinals equipped with zero* 0*, ordinal addition* $+$ *and the standard order* $\leq$ *on them constitute a derivation monoid. Note* $+$ *is not strictly monotonic in its first argument, e.g.* 0 *is smaller than* 1 *but* $0+\omega = \omega = 1+\omega$*.*
- *The* length *measure is obtained by assigning the ordinal* 1 *to all steps. An infinite reduction will then have measure* $\top$*, not* ω*. This is because to represent an infinite reduction, we need to employ at least one* $\to^\infty$*-step, making the whole reduction have measure* $\top$*. (Note the measure is independent of 'what infinite part' is represented by a* $\to^\infty$*-step.)*

The above slightly generalises [8, Definition 4] motivated by the desire and need to use ordinal measures. We assume $+$ to be strictly monotonic only in its 2nd argument in a derivation monoid, whereas in [8] strict monotonicity in both arguments was assumed. Moreover, we measure reductions from tail to head whereas in [8] they were measured from head to tail. The latter difference is only apparent as one can always transform to the other direction by using $\lambda xy.y+x$ instead of $+$. The reason for changing it nonetheless is that it allows to keep the standard ordinal operations; cf. Example 5. With this, things carry over *verbatim*:

Definition 3 ([8])**.** $\to$ *has* peak random descent *if* $a\ {}_n^{*}\!\!\leftarrow \cdot \to_\mu^\circledast b$ *with* a *in normal form implies* $a\ {}_{n'}^{*}\!\!\leftarrow b$ *with* $n' + \mu = n$*,*[4] *and* $\to$ *is* ordered *locally confluent (or* ordered *weak Church–Rosser; OWCR), if* $a\ {}_n\!\!\leftarrow \cdot \to_m b$ *implies* $a \to_{\mu'}^\circledast \cdot\ {}_{n'}^{*}\!\!\leftarrow b$ *with* $n'+m \leq \mu'+n$*, for some derivation monoid.*[5]

[2] There it is called *uniform normalisation*, which in hindsight seems not the most uniform way of naming.
[3] By adjoining $\top$ to a derivation monoid it is no longer strict in its second argument.
[4] The condition $n' + \mu = n$ implicitly captures that μ be finite, i.e. implicitly excludes infinite right legs.
[5] OWCR $\not\Rightarrow$ WCR. E.g. $b \leftarrow b \leftarrow a \to c \to c$ is OWCR since both $b \to_\top^\infty c$ and $c \to_\top^\infty b$ as b and c are looping.

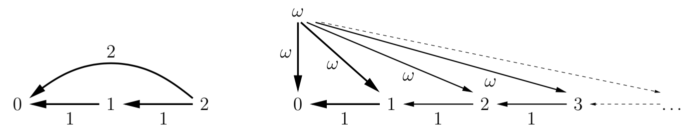


Figure 2: Measures obtained for rewrite systems of Example 1 (left) and Example 6.3 (right)

In a directed acyclic graph weights can be assigned to edges such that all paths from one node to another in it have the same weight, by topological sorting. This key idea of the proof of Lemma 1 (and thereby of this note) is illustrated in Example 6 and Figure 2.

Lemma 1. *$\to$ is uniformly complete iff $\to$ has peak random descent.*

Proof. For the only–if-direction first note that the assumption implies that all objects convertible to some normal form are complete. Thus $\to^+$ is a well-founded order on them, and we will exploit it to define a measure on the steps in conversions to normal form. We measure steps in the derivation monoid of the ordinals with 0 and ordinal addition $+$, extended with a top $\top$.

We construct measuring functions both for steps and objects, with the measure of an object being based on the measures of all its reductions to normal form. We first partition the objects into those that are convertible to some normal form, and those that are not. We measure the latter, and steps between them, arbitrarily, by 1. An object a of the former is measured by the supremum of the successors of the measures of all b such that $a \to b$. This is well-defined by well-foundedness of $\to^+$. In turn, each such step $a \to b$ is measured by the ordinal γ such that $\beta + \gamma = \alpha$, where α, β are the measures of a, b. γ exists and is non-0 per construction.

We claim peak random descent then holds. Note that it suffices to verify it for *finite* peaks $a \;{}_{\alpha}^{*}\!\leftarrow \cdot \to_{\beta}^{*} b$ with a in normal form; a right leg containing an $\to^\infty$-step would contradict uniform termination. We prove that then $\alpha = \gamma + \beta$ with γ the measure of b, by induction on the length of the peak, distinguishing cases on the direction of its last step. For the empty peak, it is trivial as normal forms have measure 0. Otherwise, $a \;{}_{\alpha'}^{*}\!\leftarrow \cdot \to_{\beta'}^{*} b' \leftrightarrow b$, where $\alpha' = \gamma' + \beta'$ holds with γ' the measure of b' by the induction hypothesis. Let δ be the measure of $b' \leftrightarrow b$. If $b' \to b$, then $\alpha = \alpha' = \gamma' + \beta' =^{(\dagger)} (\gamma + \delta) + \beta' = \gamma + (\delta + \beta') = \gamma + \beta$ where $(\dagger)$ holds since γ and γ' are the measures of b and b' and δ the measure of $b' \to b$ so $\gamma' = \gamma + \delta$. If $b' \leftarrow b$, then $\alpha = \alpha' + \delta = (\gamma' + \beta') + \delta = (\gamma' + \beta) + \delta =^{(*)} (\gamma' + \delta) + \beta = \gamma + \beta$, using for $(*)$ that this case can only happen when $\beta = 0 = \beta'$ (while constructing the left leg of the peak, its right leg is empty).

For the if-direction it suffices by Proposition 1 to show uniform termination and the normal form property. To show the former it suffices by Remark 1 to note that if $a \to b$ and b is terminating, the (finite) measure of the step and the reduction to normal form is, by peak random descent, an upper bound on the measures of the reductions from a. The latter follows by induction on the number of peaks in the conversion to normal form, cf. [8, p. 32:3]. $\square$

Example 6. *1. The measures for Example 1 are displayed on the left in Figure 2. Since c is a normal form its measure is 0. Since c is the only single-step reduct of b, the measure of b is the successor of the measure of c, i.e. 1, so the step $b \to c$ has measure 1 as well. Finally, both c and b being single-step reducts of a, the measure of a is the supremum of $\{0+1, 1+1\}$, i.e. 2, so $a \to b$ has measure 1 and $a \to c$ measure 2;*

2. It is easy to see that in Example 2 objects are measured by their number of inversions *and that all steps have measure 1, e.g. cba has 3 inversions and indeed requires 3 steps to sort;*

3. *The measures for the rewrite system having steps $a \to b_i$ and $b_{i+1} \to b_i$ for $i \in \mathbb{N}$, are displayed on the right in Figure 2. Proceeding as in the previous item, the only interesting thing to note is that the system is not* finitely branching *(FB). Accordingly a has measure the supremum of $\{i+1 \mid i \in \mathbb{N}\}$, i.e. ω, which thus is the measure of each step from it too.*

Remark 2. 1. *The proof of Lemma 1 uses that for peaks $\beta = 0 = \beta'$ in the case of $b' \leftarrow b$. For arbitrary conversions instead of peaks, this need not hold, and indeed the construction breaks down. To see this, consider the rewrite system $>$ on the ordinals up to and including ω. Proceeding as in the proof of the lemma, the objects and steps of the conversion $0 < 1 > 0 < \omega$ are measured as $0_0 \, {}_1{<} \, 1_1 >_1 0_0 <_\omega \omega_\omega$. But the measure of the backward steps in the conversion is $1 + \omega = \omega$ which is different from the sum $\omega + 1$ of the measures ω of the object ω and 1 of the forward step(s) in the conversion.*

2. *Ordinals serve to deal with systems as in Example 6.3 that are* not *finitely branching. For systems that* are *finitely branching (FB), the natural numbers suffice in the proof of Lemma 1 since then the* supremum *is the* maximum. *For* commutative conversion *monoids [8] such as the natural numbers with zero and addition, the proof generalises from peaks to conversions since then $(*)$ in the proof holds unrestrictedly.*

3 Local characterisation of uniform completeness

Lemma 2. *$\to$ has peak random descent iff $\to$ is ordered locally confluent.*

Proof. First, observe that $\to$ is ordered locally confluent iff it is *ordered* confluent, i.e. $a \; {}_n^*{\leftarrow} \cdot \to_\mu^\circledast b$ *implies $a \to_{\mu'}^\circledast \cdot \, {}_{n'}^*{\leftarrow} \; b$ with $n' + \mu \leq \mu' + n$* [8, Definition 12]. This follows from (the proof of) [8, Lemma 18], instantiating both rewrite systems with $\to$, and noting that strictness of monotonicty of $+$ was only used in its second argument (to get, for our conventions, $k' < k' + m_1$).

We show $\to$ is ordered confluent iff it has peak random descent. For the only–if-direction, ordered confluence for $a \; {}_n^*{\leftarrow} \cdot \to_\mu^\circledast b$ with a in normal form implies $a \; {}_{n'}^*{\leftarrow} \; b$ with $n' + \mu \leq n$, since a only allows the empty reduction with measure $\perp$. Applying ordered confluence to the converse of the resulting peak, comprising two reductions to normal form a, yields conversely that $n \leq n' + \mu$, hence $n' + \mu = n$. For the if-direction we distinguish cases on whether a in the peak $a \; {}_n^*{\leftarrow} \cdot \to_\mu^\circledast b$ is normalising or not. If it is, say $a \to_{m'}^* a'$ with a' in normal form, then by peak random descent for $a' \; {}_{m'+n}^*{\leftarrow} \cdot \to_\mu^\circledast b$ we have $a' \; {}_{n'}^*{\leftarrow} \; b$ with $m' + n = n' + \mu$, as desired. Otherwise, we conclude from $a \to_\top^\infty b$. ☐

Remark 3. *Whereas the first part of the proof of Lemma 2 is a special case of a commutation lemma [8, Lemma 18], the second half is not; that explicitly exploits that extending a reduction by further steps yields such a reduction again; this fails in the commutation case.*

By the two lemmata we conclude to our main result and method to establish completeness.

Theorem 1. *$\to$ is uniformly complete iff $\to$ is ordered locally confluent.*

Corollary 2. *$\to$ is complete iff $\to$ is ordered locally confluent and normalising.*

Example 7. *Consider the term rewrite rule for associativity:[6] $\varrho(x,y,z) : xyz \to x(yz)$. It is linear and it has a single critical peak which may be completed into a local confluence diagram with legs $xyzw \to x(yz)w \to x(yzw) \to x(y(zw))$ and $xyzw \to xy(zw) \to x(y(zw))$. To show*

[6] In applicative notation, using association to the left for the implicit infix application symbol @.

OWCR, observe the length measure does not work as the legs have different lengths. Measuring a step contracting $\varrho(t,s,r)$ by twice the number of leaves of t does: both legs then have the same measure: $2n+2n+2m = 2(n+m)+2n$ with n,m the number of leaves of t, s. For non-overlapping peaks ordered local confluence follows from that counting the number of leaves in a term yields a model, i.e. is invariant under ϱ. Since the bullet function of [7, Definition 32] induces a normalising strategy [7, Lemma 35(Extensive)] $\to$ is complete by the corollary.

An algebraic way of defining the measure may be obtained by employing proof terms [10, Chapter 8] to represent reductions resulting, e.g., in representing the legs of the diagram as $\varrho(x,y,z)w \cdot (\varrho(x,yz,w) \cdot x\varrho(y,z,w))$ and $\varrho(xy,z,w) \cdot \varrho(x,y,zw)$. Then the measure is defined by a 2-algebra*, i.e. an algebra for proof terms, building on a 1-algebra, i.e. an algebra on terms. In the 1-algebra, computing the number of leaves, we interpret variables by assigning 1 to them and interpret @ as addition. The 2-algebra, computing the sum of the numbers of leaves in the first argument of each ϱ-redex contracted in a reduction, builds on that by interpreting variables as 0, @ and $\cdot$ as addition, and ϱ as the 1-value of its first argument (a term).*

Example 8. *The rewrite systems in [6, Example 8] are trivially WN and locally Dyck [8, Definition 16] for the length measure, hence OWCR by [8, Theorem 19], so complete by Corollary 2. (Local Dyckness* always *works for complete rewrite systems that are FB; cf. Remark 2.2.)*

4 Conclusion

We have given an alternative complete method for establishing completeness. As also the classical method is complete it's a matter of taste and tool-support which one one prefers. It should be interesting to find a (direct) measure suitable for simply typed $\lambda\beta$ (cf. Example 4).

Acknowledgments I thank the reviewers for their feedback, despite me having uploaded an old draft instead of the intended submission, for which I apologise.

References

[1] A. Church. *The Calculi of Lambda-Conversion*. PUP, 1941.

[2] A. Geser, D. Hofbauer, and J. Waldmann. Sparse tiling through overlap closures for termination of string rewriting. In *FSCD 2019*, volume 131 of *LIPIcs*, pages 21:1–21:21. Schloss Dagstuhl, 2019.

[3] Z. Khasidashvili, M. Ogawa, and V. van Oostrom. Uniform normalisation beyond orthogonality. In *RTA 2001*, volume 2051 of *LNCS*, pages 122–136. Springer, 2001.

[4] M.H.A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[5] V. van Oostrom. Random descent. In *RTA 2007*, volume 4533 of *LNCS*, pages 314–328. Springer, 2007.

[6] V. van Oostrom. Confluence by decreasing diagrams; converted. In *RTA 2008*, volume 5117 of *LNCS*, pages 306–320. Springer, 2008.

[7] V. van Oostrom. Z; syntax-free developments. In *FSCD 2021*, volume 195 of *LIPIcs*, pages 24:1–24:22. Schloss Dagstuhl, 2021.

[8] V. van Oostrom and Y. Toyama. Normalisation by Random Descent. In *FSCD 2016*, volume 52 of *LIPIcs*, pages 32:1–32:18. Schloss Dagstuhl, 2016.

[9] G.-C. Rota. Combinatorics, representation theory and invariant theory. In *Indiscrete Thoughts*, pages 39–54. Birkhäuser, 1997.

[10] Terese. *Term Rewriting Systems*, volume 55 of *CTTCS*. CUP, 2003.

Confluence by Higher-Order Multi–One Critical pairs with an application to the Functional Machine Calculus

Willem Heijltjes and Vincent van Oostrom*

Department of Computer Science, University of Bath, United Kingdom
wbh22@bath.ac.uk, vvo21@bath.ac.uk

Abstract

The functional machine calculus (FMC) is a model of higher-order computation with effects, and is known to be confluent. Here we re-prove confluence of the FMC via higher-order term rewriting, embedding the FMC in a 3^{rd}-order PRS. Our main contribution is a higher-order version of the critical-pair-criterion that was developed by Okui for first-order TRSs, requiring all multi–one critical peaks to be many–multi joinable.

1 The Functional Machine Calculus

The *Functional Machine Calculus* (FMC) is a model of higher-order computation with effects [1]. It generalizes the λ-calculus and is known to preserve its main properties of confluence and simply-typed termination, while it encodes *reader/writer effects* (state, I/O, probabilities, nondeterminism) and strategies including call–by–name, call–by–value, and call–by–push–value [3]. In this section we recapitulate the FMC in its traditional presentation. In Section 2 we show how it can be embedded in a 3^{rd}-order positional pattern rewrite system. Via this embedding confluence of the FMC is then regained as an *instance* of a critical pair criterion for positional PRSs (Definition 2), generalising Okui's criterion for TRSs [5], as shown in Section 3.

The intuition for the FMC is of λ-terms as instruction sequences for a simple stack machine. Application $M\,N$, written $[N].\,M$, pushes N to the stack and continues with M; abstraction $\lambda x.\,M$, written $\langle x\rangle.\,M$, pops a term N and continues with $\{N/x\}M$ (the substitution of N for x in M). The FMC then consists of two generalizations. One, to multiple stacks, indexed by *locations* a, b, $c, \ldots$ in which application and abstraction are parameterized, $[N]a.\,M$ and $a\langle x\rangle.\,M$. As well as the main stack, these model input and output streams, memory cells, and random generators. Two, with the empty sequence $\star$ and *sequential composition*, implemented by making the variable construct a prefix $x.\,M$; this gives control over evaluation behaviour and models strategies. Both generalizations have interesting consequences for reduction. First, a redex consists of an application and abstraction at the same location, $[N]a \ldots a\langle x\rangle.\,M$, possibly with operations on other locations in between. Second, to substitute N for x in $x.\,M$ involves sequential composition $N\,;M$.

Definition 1. FMC-terms *are given by the following grammar, where $a\langle x\rangle.\,M$ binds x in M, and considered modulo α-equivalence. (Trailing $.\star$ may be omitted.)*

$$M,\,N,\,P \quad ::= \quad \star \;\mid\; x.\,M \;\mid\; [N]a.\,M \;\mid\; a\langle x\rangle.\,M$$

We define β-reduction by the rewrite rule schema below (closed under all contexts)

$$[N]a.\,H.\,a\langle x\rangle.\,M \;\rightarrow\; H.\,\{N/x\}M \qquad (a \notin \mathsf{loc}(H),\; \mathsf{bv}(H) \cap \mathsf{fv}(N) = \varnothing)$$

*Supported by EPSRC Project EP/R029121/1 Typed lambda-calculi with sharing and unsharing.

where H *is a* head context *with* binding variables $\mathsf{bv}(H)$ *and* locations $\mathsf{loc}(H)$ *as defined below, writing* $H.\,M$ *for* $H\{M\}$ *(*H *with the hole* $\{\}$ *replaced by* M*).*

$$H \;::=\; \{\} \mid [N]a.\,H \mid a\langle x\rangle.\,H \qquad \begin{aligned}\mathsf{bv}(\{\}) &= \varnothing\\ \mathsf{bv}([M]a.\,H) &= \mathsf{bv}(H)\\ \mathsf{bv}(a\langle x\rangle.\,H) &= \mathsf{bv}(H)\cup\{x\}\end{aligned} \qquad \begin{aligned}\mathsf{loc}(\{\}) &= \varnothing\\ \mathsf{loc}([M]a.\,H) &= \mathsf{loc}(H)\cup\{a\}\\ \mathsf{loc}(a\langle x\rangle.\,H) &= \mathsf{loc}(H)\cup\{a\}\end{aligned}$$

Composition $N\,;M$ *and substitution* $\{M/x\}N$ *are capture-avoiding, and are as follows.*

$$\begin{aligned}\star\,;M &= M & [P]a.\,N\,;M &= [P]a.\,(N\,;M)\\ x.\,N\,;M &= x.\,(N\,;M) & a\langle y\rangle.\,N\,;M &= a\langle y\rangle.\,(N\,;M) \quad (y\notin\mathsf{fv}(M))\end{aligned}$$

$$\begin{aligned}\{P/x\}\star &= \star & \{P/x\}[N]a.\,M &= [\{P/x\}N]a.\,\{P/x\}M\\ \{P/x\}x.\,M &= P\,;\{P/x\}M & \{P/x\}a\langle x\rangle.\,M &= a\langle x\rangle.\,M\\ \{P/x\}y.\,M &= y.\,\{P/x\}M \quad (x\neq y) & \{P/x\}a\langle y\rangle.\,M &= a\langle y\rangle.\,\{P/x\}M \quad (y\notin\mathsf{fv}(P))\end{aligned}$$

The pure λ-calculus may be embedded in the FMC by choosing a *main* location λ, omitted from terms for compactness, and defining $\lambda x.M = \langle x\rangle.\,M$ and $M\,N = [N].\,M$.

Example 1. *To model global store, a cell is a dedicated location* a *with* lookup $!a$ *encoded by* $a\langle x\rangle.\,[x]a.\,x$ *and* update $N := a\,;M$ *by* $a\langle\_\rangle.\,[N]a.\,M$ *(where* $\_$ *is a non-binding variable). The following example term stores* $\lambda f.\,f\,(f\,3)$ *to the cell* a*, and then retrieves it to call it on* $\lambda y.\,y{+}1$*. Overall, it should update* a *and return* 5*, which FMC reduction indeed exposes. (Underlining indicates a redex, and colours trace subterms through translations and reductions.)*

$$\begin{aligned} a := (\lambda f.\,f\,(f\,3))\,;!a\,(\lambda y.\,y+1) \;&=\; a\langle\_\rangle.\,\underline{[\langle f\rangle.\,[[3].\,f].\,f]a}.\,[\langle y\rangle.\,[y].\,[1].\,+].\,\underline{a\langle x\rangle}.\,[x]a.\,x\\ &\rightharpoonup\; a\langle\_\rangle.\,\underline{[\langle y\rangle.\,[y].\,[1].\,+]}.\,[\langle f\rangle.\,[[3].\,f].\,f]a.\,\underline{\langle f\rangle}.\,[[3].\,f].\,f\\ &\rightharpoonup\; a\langle\_\rangle.\,[\langle f\rangle.\,[[3].\,f].\,f]a.\,[[3].\,\langle y\rangle.\,[y].\,[1].\,+].\,\langle y\rangle.\,[y].\,[1].\,+\\ &\twoheadrightarrow\; a\langle\_\rangle.\,[\langle f\rangle.\,[[3].\,f].\,f]a.\,5\\ &=\; a := (\lambda f.\,f\,(f\,3))\,;5\end{aligned}$$

2 Embedding the FMC in a PRS

We show the FMC can be embedded in a 3[rd]-order *pattern rewrite system* (PRS), with which we assume familiarity [4, 9]. Since we will build on it below, we revisit the standard embedding of the pure λ-calculus in a 2[nd]-order PRS ([4, Example 3.4],[9, Examples 11.2.6(i),11.2.22(ii)]).

Example 2. *The PRS* $\mathcal{L}am$ *has a single base type* term*, two simply typed constants for abstraction and application:* $\mathsf{lam} : (\mathsf{term}\to\mathsf{term})\to\mathsf{term}$ *and* $\mathsf{app} : \mathsf{term}\to\mathsf{term}\to\mathsf{term}$*, and rules:*

$$\begin{aligned}\mathsf{beta} \;&:\; \lambda FS.\mathsf{app}(\mathsf{lam}\lambda x.F(x),S) \;\to\; \lambda FS.F(S)\\ \mathsf{eta} \;&:\; \lambda S.\mathsf{lam}(\lambda x.\mathsf{app}(S,x)) \;\to\; \lambda S.S\end{aligned}$$

with variables $x:\mathsf{term}$*,* $F:\mathsf{term}\to\mathsf{term}$ *and* $S:\mathsf{term}$*, and rules, which are symbols in our setting having the type of their lhs / rhs,* $\mathsf{beta} : (\mathsf{term}\to\mathsf{term})\to\mathsf{term}\to\mathsf{term}$*, and* $\mathsf{eta} : \mathsf{term}\to\mathsf{term}$*.*

Objects The objects of a PRS are simply typed λ-terms modulo $\alpha\beta\eta$ for a collection of *base* types, and a signature of *symbols*. We refer to the simply typed $\lambda\alpha\beta\eta$-calculus as the *substitution* calculus of PRSs as it brings about the standard notions of matching, substitution and occurrence [8, 6]. We assume λ-terms to be in η-expanded form ([4, p. 5],[9, Convention 11.2.12]). *Terms* then are λ-terms also in β-normal form, serving as representatives (unique up to α) of $\alpha\beta\eta$-equivalence classes. The *parameter passing* of rewrite rules is brought about by the substitution calculus, *matching* by β-expansion and *substitution* by β-reduction. To separate the *replacement* aspect of rewrite rules from their parameter passing aspect [9, Definition 11.2.25(iv)], rewrite rules are closed. To facilitate defining *occurrences* below, we overline a subterm of a λ-term to denote the λ-term (recursively) obtained by removing the overlining, and if the subterm is a β-redex then contracting it and overlining the created β-redexes.

In [2, Lemma 2] we established that for first-order term rewriting there is a perfect rapport between the *inductive* and *geometric* views of the notion of *occurrence*. We consider the higher-order case: in the inductive view an occurrence of a *pattern* π in a λ-term t then is a β*-expansion* of t to a λ-term $(\lambda\, x.s)\,\pi$ (cf. [8, Definition 2.9]), and in the geometric view a *pat* P is a certain *subset* of the positions of the tree [4, p. 5] of t (cf. [9, Proposition 8.6.25]). To make the rapport perfect, we restrict ourselves to occurrences of *patterns* [4, Definition 3.1] that are *rule-patterns* [9, Definition 11.2.18(ii)], *local* [7, Footnote 4], and moreover such that the free variables are in pre-order and the parameters in outside-in order; these are *positional* patterns:

Definition 2 (Inductive view). *A* positional pattern π *is a closed λ-term of shape* $\lambda\,\boldsymbol{F}.f(\boldsymbol{t})$ *such that (*head-defined*) f is a function symbol and $f(\boldsymbol{t})$ is of base type; (*linear*) π is linear in $\boldsymbol{F}$, each F_i occurs once; and (*fully-extended*) each $F \in \boldsymbol{F}$ occurs in π as $F(\boldsymbol{x})$ where x is the list of (η-expansions of) variables that are bound above F in $f(\boldsymbol{t})$, in outside-in order. To avoid clutter we may drop the initial binders $\boldsymbol{F}$ of π. We incongruously refer to such an F as a* free variable *of π and to its arguments $\boldsymbol{x}$ as its* parameters*. A rule / PRS is* <u>positional</u> *if its lhs is / rules are. If for a vector $\boldsymbol{\pi}$ of positional patterns and λ-term t, we have $\overline{(\lambda\,\boldsymbol{F}.s)\,\boldsymbol{\pi}} = t$ with s linear in $\boldsymbol{F}$, we speak of a* multipattern $\boldsymbol{\pi}$ in t. *They are taken up to permutation of $\boldsymbol{\pi}$, $\boldsymbol{F}$.*

Definition 3 (Geometric view). *A* pat *in a λ-term t is a non-empty set P of positions in the tree*[1] *of t such that (*convex*) if $p, q \in P$ then all positions on the path between p and q are in P [2, Footnote 4]; (*rigid*) if $t(p)$ is a variable and $p \in P$, then it is bound by a λ-abstraction at a position in P; (*base-fringe*) $t_{|p}$ is of base type for p the root of P or a child not in P of a position in P; and (*normal*) if $t(p)$ is an application and $p \in P$, then its left child is not the position of a λ-abstraction. A* multipat *is a vector $\boldsymbol{P}$ of pairwise disjoint pats in t.*

Example 3. *For examples of patterns see [9, Example 11.2.19]. The lhs of* beta *is a positional pattern. It would not be so anymore when swapping its initial binders from $\lambda\, FS$ into $\lambda\, SF$ (pre-order violated). The lhs of* eta *is a pattern, but is not positional (full-extendedness violated).*

For π the lhs of beta*, we have $\{11, 111, 1111, 1112, 11121, 11122\}$ is a pat; $11, 111, 1111$ are the positions from its root 11 toward the head symbol* app*, 11121 the position of* abs*, and 11122 that of $\lambda\, x$. This is the* greatest *pat in π, its* internal *pat $\mathring{\pi}$. The only other pat in π is $\{1112, 11121, 11122\}$ corresponding to* $\mathsf{lam}\lambda\, x.F(x)$*. For instance, $\{1112, 11121\}$ is not a pat, since the subterm $\lambda\, x.F(x)$ at position 11122 is not of base type violating (base-fringe), and $\{112\}$ is not a pat since (rigid) is violated by S being a free variable. For TRSs, a pat coincides with a non-empty convex set of function symbol positions as in [2].*

Multipatterns and multipats can be ordered by *refinement* $\sqsubseteq$. These orders correspond and will allow us to state the notion of critical peak in lattice-theoretic terms [2].

Definition 4. $(\lambda\,\boldsymbol{G}.(\overline{(\lambda\,\boldsymbol{F}.s)\,\boldsymbol{u}}))\,\boldsymbol{\pi} \sqsubseteq \overline{(\lambda\,\boldsymbol{G}.((\lambda\,\boldsymbol{F}.s)\,\boldsymbol{u}))\,\boldsymbol{\pi}}$ *if both sides are multipatterns and s, $\boldsymbol{u}$ are linear in $\boldsymbol{F}$, $\boldsymbol{G}$. For multipats, $\boldsymbol{Q} \sqsubseteq \boldsymbol{P}$ if each pat $Q \in \boldsymbol{Q}$ is a subset of a pat $P \in \boldsymbol{P}$.*

[1] We employ $t_{|p}$ / $t(p)$ to denote the subterm / symbol *at* position p in t ([4, p. 5] uses t/p for the former).

Example 4. *We have* $\{\{2,21\},\{222,2221\}\} \sqsubseteq \{\{2,21,22,221,222,2221\}\}$ *for multipats in* $f(g(h(i(a))))$*. Likewise* $(\lambda\, XY.f(X(h(Y(a))))\,(\lambda\, z.g(z))\,(\lambda\, z.i(z)) \sqsubseteq (\lambda\, Z.f(Z(a)))\,\lambda\, z.g(h(i(z)))$ *for multipatterns as witnessed by* $(\lambda\, XY.(\lambda\, Z.f(Z(a)))\,(\lambda\, z.X(h(Y(z)))))\,(\lambda\, z.g(z))\,(\lambda\, z.i(z))$.

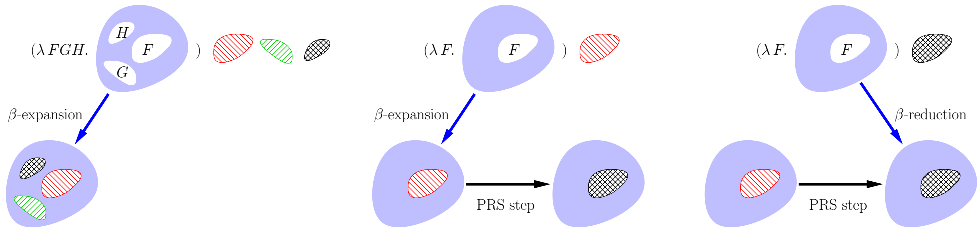


Figure 1: Carving out multipat from term by β-expanding into multipattern (left), and step for PRS rule $\ell \to r$ via matching (β-expansion; middle) and substitution (β-reduction; right)

Lemma 1. *Refinement* $\sqsubseteq$ *on multipats / multipatterns of a* λ*-term is a finite distributive lattice. Multipatterns and multipats w.r.t. their respective notions of refinement* $\sqsubseteq$*, are isomorphic.*

Proof idea. By extending the proof of-[2, Lemma 2] to positional PRSs. The isomorphism between multipats and multipatterns is illustrated in Figure 1; for any multipat $\boldsymbol{P}$ in a λ-term t a multipattern $\boldsymbol{\pi}$ may be *carved out* from t in that $\overline{(\lambda\, \boldsymbol{F}.s)\,\boldsymbol{\pi}} = t$ for some s linear in $\boldsymbol{F}$ such that the set of *internal* positions of the $\boldsymbol{\pi}$ in it *trace* [9] to the $\boldsymbol{P}$ in t, and vice versa. ☐

Steps The *steps* of a PRS are *terms* over the signature extended with rules [9, Chapter 8].

Definition 5. *A* multistep *of a PRS* $\mathcal{P}$ *is a term over its signature extended with its rule symbols. This induces a rewrite system* $\multimap_{\mathcal{P}}$ *having terms as objects, multisteps as steps, with source / target maps obtained substituting the lhs / rhs for the rule symbol [8, 6]; cf. Figure 1 (middle,right). Requiring to have* one *rule in a multistep yields* steps $\to_{\mathcal{P}}$.

Example 5. $\mathsf{abs}(\lambda\, y.\mathsf{beta}(\lambda\, x.\mathsf{app}(x,x),y)))$ *and* $\mathsf{eta}(\mathsf{abs}(\lambda\, x.\mathsf{app}(x,x)))$ *are* Lam*-steps. Despite being intensionally distinct, they are extensionally the same as they have the same sources* $\mathsf{abs}(\lambda\, y.\overline{(\lambda\, FS.\mathsf{app}(\mathsf{lam}\lambda\, x.F(x),S))}\,(\lambda\, x.\mathsf{app}(x,x),y))) = \mathsf{abs}(\lambda\, y.\underline{\mathsf{app}(\mathsf{abs}(\lambda\, x.\mathsf{app}(x,x)),y)}) = (\lambda\, S.\mathsf{lam}(\lambda\, x.\mathsf{app}(S,x)))\,\underline{(\mathsf{abs}(\lambda\, x.\mathsf{app}(x,x)))}$ *and targets* $\mathsf{abs}(\lambda\, y.\overline{(\lambda\, FS.F(S))}\,(\lambda\, x.\mathsf{app}(x,x),y))) = \mathsf{abs}(\lambda\, y.\mathsf{app}(y,y)) = \overline{(\lambda\, S.S)}\,(\mathsf{abs}(\lambda\, x.\mathsf{app}(x,x)))$.

Multisteps render traditional redex-orthogonality-talk obsolete [2]; redexes are orthogonal *because* there is a multistep contracting them. Note $\to_{\mathcal{P}} \subseteq \multimap_{\mathcal{P}} \subseteq \twoheadrightarrow_{\mathcal{P}}$ [9, Lemma 11.6.24(ii)].

The FMC as fragment of a PRS The untyped λ-calculus is embedded in a *fragment* of the 2[nd]-order PRS $\mathcal{Lam}$, namely in terms where all variables are of type term. We show the same holds for the FMC: its terms are embedded as a fragment of a 3[rd]-order PRS $\mathcal{FMC}$. The embedding hinges on that although the FMC (Definition 1) has a non-standard notion of substitution, that may be *represented* by PRS substitution by replacing each $\star$ by a variable χ, so that *composition* with N in the FMC is represented in $\mathcal{FMC}$ as substitution of N for χ.

Definition 6. *The PRS* $\mathcal{FMC}$ *has a signature comprising for every location* a*, symbols* $\mathsf{lam}_a : ((\mathsf{term} \to \mathsf{term}) \to \mathsf{term}) \to \mathsf{term}$ *and* $\mathsf{app}_a : \mathsf{term} \to (\mathsf{term} \to \mathsf{term}) \to \mathsf{term}$*, and rewrite rule schema:*

$$\mathsf{beta}_H \quad : \quad \lambda\, M\boldsymbol{P}N.\mathsf{app}_a(H[\mathsf{lam}_a(\lambda\, x.M(\boldsymbol{x},x))],N) \quad \to \quad \lambda\, M\boldsymbol{P}N.H[M(\boldsymbol{x},N)]$$

where N, $\boldsymbol{x}$, and x all have type $\mathsf{term} \to \mathsf{term}$ *(not η-expanded to avoid clutter) and H ranges over* contexts*, compositions of basic contexts with the empty context $\square$, with a* basic context *being of shape either* $\mathsf{app}_b(\square, P(\boldsymbol{x}))$ *or* $\mathsf{lam}_b(\lambda\, x.\square)$*, for any location b distinct from a, and each $P \in \boldsymbol{P}$ a fresh free variable having as parameters the variables bound by the contexts above it.*

Terms of the FMC are represented by *spines*, $\mathcal{FMC}$-terms $\lambda\,\chi.S$ of type $\mathsf{term} \to \mathsf{term}$ with:

$$S \quad ::= \quad \chi \mid x\,S \mid \mathsf{app}_a(S, \lambda\,\chi.S) \mid \mathsf{lam}_a(\lambda\, x.S)$$

where χ is the *unique* variable of type term. We embed an FMC term M as $\lambda\,\chi.\langle M\rangle$ and show this fragment of $\mathcal{FMC}$ is well-behaved, where $\langle\,\rangle$ maps the FMC constructs as follows: (i) $\star$ is mapped by $\langle\,\rangle$ to χ, that is, to the coccyx of a spine; (ii) $x.\,M$ is mapped to $x\langle M\rangle$, that is, to the application of x to the embedding of M; (iii) $[N]a.\,M$ is mapped to $\mathsf{app}_a(\langle M\rangle, \lambda\,\chi.\langle N\rangle)$; and (iv) $a\langle x\rangle.\,M$ is mapped to $\mathsf{lam}_a(\lambda\, x.\langle M\rangle)$.

Lemma 2. *Embedding the FMC in the $\lambda\,\chi.S$-fragment yields a bisimulation for $\rightharpoonup$ and $\to_{\mathsf{beta}_H}$.*

3 A Multi–One Critical Pair Criterion for the FMC

We generalise the critical pair criterion for confluence introduced in [5] from left-linear TRSs to positional PRSs to obtain confluence of $\mathcal{FMC}$, and hence (Lemma 2) of its $\lambda\,\chi.S$-fragment.

Definition 7. *Multipatterns ς and ζ in term t are* overlapping *if $\varsigma \sqcap \zeta \neq \bot$, where $\sqcap$ denotes the meet w.r.t. refinement $\sqsubseteq$ and $\bot$ the least element (t). The overlap is* critical *if moreover $\varsigma \sqcup \zeta = \overline{(\lambda\,F.\hat{F})}\,t$ with $\hat{F}$ the η-expansion of F. This extends to peaks ${}_{\Phi}\!\leftarrow\!\circ\!\!- \; t \to_{\Psi}$ of multisteps $\Phi = \overline{(\lambda\,\boldsymbol{F}.s)}\,\boldsymbol{\varrho}$ and $\Psi = \overline{(\lambda\,\boldsymbol{G}.u)}\,\boldsymbol{\theta}$ for rules $\boldsymbol{\varrho} : \boldsymbol{\ell} \to \boldsymbol{r}$ and $\boldsymbol{\theta} : \boldsymbol{g} \to \boldsymbol{d}$, via their multipatterns $(\lambda\,\boldsymbol{F}.s)\,\boldsymbol{\ell}$ and $(\lambda\,\boldsymbol{G}.u)\,\boldsymbol{g}$. If Ψ is a step, we speak of a multi–one (critical) peak.*

Example 6. *We give two multi–one critical peaks for the following TRS [5, Example 1], with our multi–one critical peaks corresponding to the critical pairs numbered (4) and (5) there:*

$$\begin{array}{ll} \alpha : \lambda\, xyz.x + (y+z) \to \lambda\, xyz.(x+y)+z \\ \gamma : \qquad\quad \lambda\, xy.x + y \to \lambda\, xy.y + x \end{array}$$

$$\lambda\, xyz.(z+y)+x \;{}_{(\lambda\, FGxyz.F(x,G(y,z)))\,\gamma\gamma}\!\leftarrow\!\circ\!\!-\; \lambda\, xyz.x+(y+z) \;\to_{(\lambda\, Hxyz.H(x,y,z))\,\alpha}\; \lambda\, xyz.(x+y)+z$$

$$\lambda\, \boldsymbol{w}.((x+y)+z)+w \;{}_{(\lambda\, FG\boldsymbol{w}.F(w,G(x,y,z)))\,\gamma\alpha}\!\leftarrow\!\circ\!\!-\; \lambda\, \boldsymbol{w}.w+(x+(y+z)) \;\to_{(\lambda\, H\boldsymbol{w}.H(w,x,y+z))\,\alpha}\; \lambda\, \boldsymbol{w}.(w+x)+(y+z)$$

where $\boldsymbol{x} = wxyz$. The first multi–one peak has $\{111\cdot\{\varepsilon,1,11\}, 111\cdot\{2,21,211\}\}$ as multipat for the left multistep and $\{111\cdot\{\varepsilon,1,11,2,21,211\}\}$ for the right.

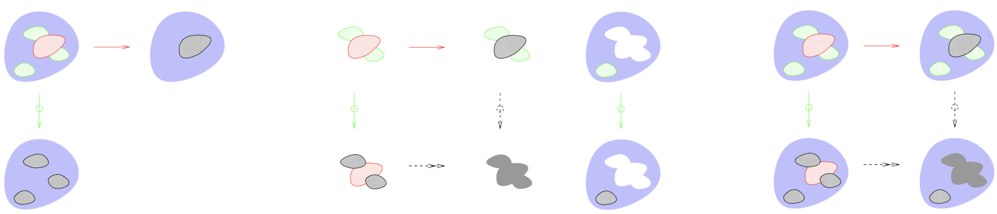


Figure 2: Illustration of proof of Lemma 3 by *splitting-off* critical multi–one peak

.

Lemma 3. *If for a positional PRS $\mathscr{P}$ every critical multi–one peak is many–multi joinable, i.e. if ${}_{\Phi}{\leftarrow\!\!\circ\!\!-} \cdot \to_{\Psi} \subseteq \twoheadrightarrow_{\mathscr{P}} \cdot {}_{\mathscr{P}}{\leftarrow\!\!\circ\!\!-}$ for Φ, Ψ critical, then multi–one peaks are many–multi joinable.*

Proof idea. Let $s \;{}_{\Phi}{\leftarrow\!\!\circ\!\!-}\; t \to_{\Psi} u$ be a multi–one peak. The geometric view, justified by Lemma 1, for the following construction is illustrated in Figure 2 where the blue blob denotes t, the green blobs the multipat of the multistep $-\!\!\circ\!\!\to_{\Phi}$, and the red blob that of the step $\to_{\Psi}$.

We may write the multipattern ς of Φ as $(\lambda\, \boldsymbol{G'G}.s')\,\boldsymbol{\ell'\ell}$, and the multipattern ζ of Ψ as $(\lambda\, F.u')\, g$, with $\boldsymbol{\ell}$ those patterns in ς overlapping the pattern g (2 green blobs in the figure overlapping the red one), and $\boldsymbol{\ell'}$ (1 green blob) the non-overlapping ones;

The join $\varsigma \sqcup \zeta$ is then of shape $\overline{(\lambda\, \boldsymbol{G'}F'.v)\,\boldsymbol{\ell'}\pi}$ with π being the minimal pattern refinable into both $\boldsymbol{\ell}$ and g. Thus, $\varsigma = (\lambda\, \boldsymbol{G'G}.\overline{(\lambda\, \boldsymbol{G'}F'.v)\,\boldsymbol{G'}s''})\,\boldsymbol{\ell'\ell}$ and $\zeta = (\lambda\, F.\overline{(\lambda\, \boldsymbol{G'}F'.v)\,\boldsymbol{\ell'}u''})\, g$ for some s'' and u'', with the multisteps Φ and Ψ obtained by replacing the left-hand sides $\boldsymbol{\ell'\ell}$ in ς and g in ζ by rule symbols, and with $\overline{(\lambda\, \boldsymbol{G'G}.(\lambda\, \boldsymbol{G'}F'.v)\,\boldsymbol{G'}s'')\,\boldsymbol{\ell'\ell}} = \varsigma \sqcup \zeta = \overline{(\lambda\, F.(\lambda\, \boldsymbol{G'}F'.v)\,\boldsymbol{\ell'}u'')\, g}$. By minimality, π is the source of the *critical* multi-one peak for multistep $\hat{\Phi}$ and step $\hat{\Psi}$ having multipatterns $(\lambda\, \boldsymbol{G}.s'')\,\boldsymbol{\ell}$ and $(\lambda\, F.u'')\, g$, which is many–multi joinable by assumption, say by valley $\twoheadrightarrow_{\hat{\Psi}'} \cdot {}_{\hat{\Phi}'}{\leftarrow\!\!\circ\!\!-}$ for reduction $\hat{\Psi}'$ and multistep $\hat{\Phi}'$. We conclude by *plugging these into context* as in Figure 2 (right), yielding the reduction $(\lambda\, \boldsymbol{G'}F'.v)\,\boldsymbol{r'}\hat{\Psi}'$ and the multistep $(\lambda\, \boldsymbol{G'}F'.v)\,\boldsymbol{\varrho'}\hat{\Phi}'$, for $\boldsymbol{r'}$ and $\boldsymbol{\varrho'}$ the right-hand sides respectively the rules, corresponding to $\boldsymbol{\ell'}$. ☐

Theorem 1. *A positional PRS is confluent if multi–one critical peaks are many–multi joinable.*

Proof. By Lemma 3 using $\to_{\mathscr{P}} \subseteq -\!\!\circ\!\!\to_{\mathscr{P}} \subseteq \twoheadrightarrow_{\mathscr{P}}$ for any positional PRS $\mathscr{P}$. ☐

Theorem 2. *FMC reduction is confluent.*

Proof. By Theorem 1 and Lemma 2 it suffices that all multi–one critical peaks of $\mathscr{FMC}$ are many–multi joinable. There are still infinitely many such peaks, but these are uniformly shown to be many–multi joinable: since in the FMC all patterns in a critical peak are on the same spine, and patterns on the spine are not replicated, the peaks are even one–multi joinable. ☐

References

[1] C. Barrett, W. Heijltjes, and G. McCusker. The functional machine calculus, 2022. To appear in Mathematical Foundations of Programming Semantics (MFPS 2022). Available at http://people.bath.ac.uk/wbh22/index.html#FMC2022.

[2] N. Hirokawa, J. Nagele, V. van Oostrom, and M. Oyamaguchi. Confluence by critical pair analysis revisited. In *CADE 27*, volume 11716 of *LNCS*, pages 319–336. Springer, 2019.

[3] P.B. Levy. *Call-by-push-value: A functional/imperative synthesis*, volume 2 of *Semantic Structures in Computation*. Springer Netherlands, 2003.

[4] R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *TCS*, 192(1):3–29, 1998.

[5] S. Okui. Simultaneous critical pairs and Church–Rosser property. In T. Nipkow, editor, *RTA-98*, volume 1379 of *LNCS*, pages 2–16. Springer, 1998.

[6] V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit, Amsterdam, March 1994.

[7] V. van Oostrom. Finite family developments. In H. Comon, editor, *RTA-97*, volume 1232 of *LNCS*, pages 308–322. Springer, 1997.

[8] V. van Oostrom and F. van Raamsdonk. Weak orthogonality implies confluence: The higher order case. In *LFCS'94*, volume 813 of *LNCS*, pages 379–392. Springer, 1994.

[9] Terese. *Term Rewriting Systems*, volume 55 of *CTTCS*. CUP, 2003.

On Confluence of Parallel-Innermost Term Rewriting*

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

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

Abstract

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

1 Introduction

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

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

Figure 1: Tree size computation in Rust

Here, the recursive calls to `left.size()` and `right.size()` can be done in parallel. In the following, we shall consider a corresponding parallel-innermost rewrite relation. Evaluation of TRSs (as a simple functional programming language) with innermost rewrite strategies in massively parallel settings such as GPUs is currently a topic of active research [14]. Confluence of parallel-innermost rewriting enters the picture in several ways: for TRSs, confluence determines whether the specific choice of rules makes a difference; moreover, confluence can be a prerequisite for applicability of program analysis techniques (e.g., for finding complexity bounds [3]).

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

Some of the results presented in this extended abstract, along with an application to automated complexity bounds analysis of parallel-innermost rewriting, can be found also in the conference paper [3].

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

2 Preliminaries

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

Definition 1 (Term Rewrite System, Innermost Rewriting)**.** *$\mathcal{T}(\Sigma, \mathcal{V})$ denotes the set of* terms *over a finite signature Σ and the set of variables $\mathcal{V}$. For a term t, the set $\mathcal{P}os(t)$ of its* positions *is given as: (a) if $t \in \mathcal{V}$, then $\mathcal{P}os(t) = \{\varepsilon\}$, and (b) if $t = f(t_1, \ldots, t_n)$, then $\mathcal{P}os(t) = \{\varepsilon\} \cup \bigcup_{1 \le i \le n} \{i\pi \mid \pi \in \mathcal{P}os(t_i)\}$. The position ε is the* root position *of term t. For $\pi \in \mathcal{P}os(t)$, $t|_\pi$ is the subterm of t at position π, and we write $t[s]_\pi$ for the term that results from t by replacing the subterm $t|_\pi$ at position π by the term s. A* substitution *σ is a partial mapping from $\mathcal{V}$ to $\mathcal{T}(\Sigma, \mathcal{V})$. The application of a substitution σ to a term t, written $t\sigma$, is defined inductively: (a) if $t \in \mathcal{V}$ and $\sigma(t)$ is defined, then $t\sigma = \sigma(t)$; (b) if $t \in \mathcal{V}$ and $\sigma(t)$ is not defined, then $t\sigma = t$; (c) if $t = f(t_1, \ldots, t_n)$, then $t\sigma = f(t_1\sigma, \ldots, t_n\sigma)$.*

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

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

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

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

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

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

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

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

3 Parallel-Innermost Rewriting

The notion of parallel-innermost rewriting dates back at least to the year 1974 [15]. Informally, in a parallel-innermost rewrite step, all innermost redexes are rewritten simultaneously. This corresponds to executing all function calls in parallel using a call-by-value strategy on a machine with unbounded parallelism [4]. In the literature [13], this strategy is also known as "max-parallel-innermost rewriting".

Definition 2 (Parallel-Innermost Rewriting [6])**.** *A term s* rewrites innermost in parallel *to t with a TRS $\mathcal{R}$, written $s \mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} t$, iff $s \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}} t$, and either (a) $s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t$ with s an innermost redex, or (b) $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, and for all $1 \leq k \leq n$ either $s_k \mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} t_k$ or $s_k = t_k$ is a normal form.*

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

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

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

4 Confluence of Parallel-Innermost Rewriting

Given a TRS $\mathcal{R}$, we wish to prove (or disprove) confluence of this relation $\mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$. Apart from intrinsic interest in confluence as an important property of a rewrite relation, we are also motivated by applications [3] of confluence proofs to finding *lower bounds* for the length of the longest derivation with $\mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ from *basic terms*, i.e., terms $f(t_1, \ldots, t_k)$ where f is a defined symbol and all t_i are constructor terms. This notion of *complexity* of a TRS $\mathcal{R}$, which is parametric in the *size* of the start term, is also known as *runtime complexity* [9].[1]

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

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

Example 3 (Confluence of $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ Does Not Imply Confluence of $\mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$)**.** *To see that we cannot prove confluence of $\mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ just by using a standard off-the-shelf tool for confluence analysis of innermost or full rewriting [5], consider the TRS $\mathcal{R} = \{\mathsf{a} \to \mathsf{f}(\mathsf{b}, \mathsf{b}), \mathsf{a} \to \mathsf{f}(\mathsf{b}, \mathsf{c}), \mathsf{b} \to \mathsf{c}, \mathsf{c} \to \mathsf{b}\}$. For this TRS, both $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ and $\to_{\mathcal{R}}$ are confluent. However, $\mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ is not confluent: we can rewrite both $\mathsf{a} \mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{b}, \mathsf{b})$ and $\mathsf{a} \mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{b}, \mathsf{c})$, yet there is no term v such that $\mathsf{f}(\mathsf{b}, \mathsf{b}) \mathrel{\|\!\xrightarrow{\mathsf{i}}}{}^{*}_{\mathcal{R}} v$ and $\mathsf{f}(\mathsf{b}, \mathsf{c}) \mathrel{\|\!\xrightarrow{\mathsf{i}}}{}^{*}_{\mathcal{R}} v$. The reason is that the only possible rewrite sequences with $\mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ from these terms are $\mathsf{f}(\mathsf{b}, \mathsf{b}) \mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{c}, \mathsf{c}) \mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{b}, \mathsf{b}) \mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \ldots$ and $\mathsf{f}(\mathsf{b}, \mathsf{c}) \mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{c}, \mathsf{b}) \mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{b}, \mathsf{c}) \mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \ldots$, with no terms in common.*

Thus, in general a confluence proof for $\to_{\mathcal{R}}$ or $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ does not imply confluence for $\mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$. Intuitively, the reason for non-confluence for $\mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ in Example 3 is the non-termination of $\mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$. We leave the following open conjecture to future work.

Conjecture 1. *Let $\mathcal{R}$ be a TRS whose innermost rewrite relation $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is terminating. Then $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is confluent iff $\mathrel{\|\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ is confluent.*

[1]The details of our approach to finding complexity bounds are outside of the scope of the present extended abstract and can be found in [3]; what matters here is that it provides an *application* for techniques to prove confluence of parallel-innermost rewriting. Thus, more powerful techniques for proving confluence of parallel-innermost rewriting potentially allow for larger applicability of techniques for finding lower bounds for runtime complexity of parallel-innermost rewriting.

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

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

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

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

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

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

Corollary 1 (Confluence of Parallel-Innermost Rewriting). *If a TRS $\mathcal{R}$ is non-overlapping, then $\mathrel{\smash{\overset{\mathsf{i}}{\twoheadrightarrow\!\!\!\!\!\!\!\shortparallel}}}_{\mathcal{R}}$ is confluent.*

Here left-linearity of $\mathcal{R}$ (i.e., in all rules $\ell \to r \in \mathcal{R}$, every variable occurs at most once in ℓ), as in Rosen's criterion for confluence of full rewriting [12], is not required.

Example 4 (Example 2 continued). *Our TRS $\mathcal{R}$ from Example 1 and Example 2 is non-overlapping and, by Corollary 1, its relation $\mathrel{\smash{\overset{\mathsf{i}}{\twoheadrightarrow\!\!\!\!\!\!\!\shortparallel}}}_{\mathcal{R}}$ is confluent.*

The reasoning behind Corollary 1 can be generalised to *arbitrary* parallel rewrite strategies where the redexes that are rewritten are fixed, such as (max-)parallel-outermost rewriting [13].

We get two follow-up questions:

1. How powerful is Corollary 1 for proving confluence of $\mathrel{\smash{\overset{\mathsf{i}}{\twoheadrightarrow\!\!\!\!\!\!\!\shortparallel}}}_{\mathcal{R}}$ in practice?
2. Can we really not do better than Corollary 1?

5 Experiments

To assess the first question, we used the implementation of the non-overlappingness check in the automated termination and complexity analysis tool AProVE [7]. To demonstrate the effectiveness of our implementation, we have considered the 663 TRSs from category `Runtime_Complexity_Innermost_Rewriting` of the Termination Problem Database (TPDB), version 11.2 [17]. The TPDB is a collection of examples used at the annual *Termination and Complexity Competition* [8, 16]. The above category of the TPDB is the benchmark collection used specifically to compare tools that infer complexity bounds for runtime complexity of *innermost rewriting*. As both the TPDB and also COPS [10], the benchmark collection used in the *Confluence Competition* [5], currently do not have a specific benchmark collection focused on parallel-innermost rewriting, we used this benchmark collection for our first experiment.[2]

[2]Our experimental data as well as all examples are available online [1].

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

However, the nature of the benchmark set also plays a significant role. As a benchmark collection for our second experiment, we downloaded the 570 unsorted unconditional TRSs of COPS.[3] While the TRSs in this subset of COPS are usually analysed for confluence of full rewriting, we analysed whether the TRSs are confluent for parallel-innermost rewriting. Our implementation determined that 115 of the 570 TRSs (about 20.2%) are non-overlapping, which implies parallel-innermost confluence. The significantly lower success rate for this benchmark set is not surprising: COPS collects TRSs that provide a challenge to confluence analysis tools, whereas the analysed subset of the TPDB contains TRSs which are interesting specifically for runtime complexity analysis.

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

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

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

A natural direction for this research would be to investigate whether other existing criteria for confluence of full rewriting can be adapted to parallel-innermost rewriting. For example, one might investigate whether a variant of the criterion by Knuth and Bendix [11] using termination and joinability of critical pairs as a criterion for confluence of full rewriting could also be applied to parallel-innermost rewriting. We leave this question to future work.

6 Conclusion

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

Acknowledgements. We thank the anonymous reviewers for many helpful remarks and suggestions.

[3] The download took place on 28 June 2022.

References

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

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

[3] Thaïs Baudon, Carsten Fuhs, and Laure Gonnord. Analysing parallel complexity of term rewriting. In *Proc. LOPSTR '22*, LNCS. Springer, 2022. To appear.

[4] Guy E. Blelloch and John Greiner. Parallelism in sequential functional languages. In *Proc. FPCA '95*, pages 226–237. ACM, 1995.

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

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

[7] 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.

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

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

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

[11] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

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

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

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

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

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

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

Checking Confluence of Rewrite Rules in Haskell

Makoto Hamana   Faustin Yao Date

Faculty of Informatics, Gunma University, Japan

Abstract

We present a tool GSOL, a confluence checker for GHC. It checks the confluence property for rewrite rules in a Haskell program by using the confluence checker SOL (Second-Order Laboratory). The Glasgow Haskell Compiler (GHC) allows the user to use rewrite rules to optimize Haskell programs in the compilation pipeline. Currently, GHC does not check the confluence of the user-defined rewrite rules. If the rewrite rules are not confluent then the optimization using these rules may produce unexpected results. Therefore, checking the confluence of rewrite rules is important. We implement GSOL using the plugin mechanism of GHC.

1 Introduction

The Glasgow Haskell Compiler (GHC) is an open source compiler and interactive environment for the functional language Haskell. It has builtin transformation rules to optimize Haskell programs during the compilation. The user can also add rewrite rules to a program to specify optimizing transformations [13]. The notion of *confluence* is one of the important properties of rewrite rules known in the theory of rewriting. Confluence guarantees the uniqueness of normal forms, which is particularly desirable in functional programming. However, GHC does not attempt to check the confluence of user-defined rewrite rules.

In this paper, we present GSOL, a GHC plugin to check the confluence of rewrite rules in a Haskell program. It uses a modified version of confluence checker, Second-Order Laboratory (SOL) [9, 10]. that can cope with higher-order polymorphic rules. To illustrate our work, we consider the following Haskell program that involves two rewrite rules.

```
module F where                          e = f 99

{-# RULES                               {-# NOINLINE f #-}
        "f/0" forall x. f x = 0         f :: Integer -> Integer
        "f/1" forall x. f x = 1         f x = 0
 #-}
```

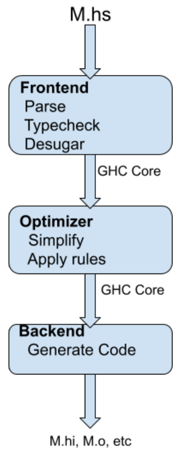
The code within the `{-# ... #-}` is called a *pragma* [1]. In the `RULES` pragma, there are two rules named `"f/0"` and `"f/1"`. If the the compiler chooses to apply the rule `"f/0"` then the expression `f 99` is rewritten to 0. If it chooses to apply the rule `"f/1"` then the expression `f 99` is rewritten to 1. Therefore, they are not confluent. But the GHC compiler *does not* notice this non-confluence.

To the best of our knowledge, there is no tool to check the confluence of rewrite rules directly from a Haskell program. In the field of term rewriting, a few confluence checkers for *higher-order* rewrite systems have been developed [16, 15, 9]. We use a higher-order modified version of the tool SOL [9, 10] to check the confluence of GHC rewrite rules. Since SOL has participated in the Higher-Order Rewriting category of the International Confluence Competition 2018 [1] and 2020 [3], and has shown that its effectiveness.

This work. In the previous works on rewrite rules for optimizations including [7, 13] confluence and termination of rewrite rules have not been checked automatically although ensuring them has been recognized as an important problem. In this work, we solve this problem by applying the result of well-established rewriting technology to the real-world functional programming language Haskell. We use an automatic confluence checker SOL to check the confluence of GHC rewrite rules in a Haskell program.

[1]The `NOINLINE` pragma instructs the compiler not to expand `f 99` by using the function definition. Without this indication, `f 99` is inlined to `0` before applying the rewrite rules, resulting that no rewrite rules are fired.

2 Background



GHC. The compilation process of a Haskell program consists of three big steps: *frontend*, *optimizer*, and *backend*. The frontend consists of parsing, type checking and the transformation into the GHC's intermediate language called *GHC Core*, implementing System F_C [14]. A GHC Core expression consists of variables, literals, abstractions, applications and variable bindings. The optimizer optimizes the GHC Core program through various transformations. The *simplifier* implements most of those transformations using a set of built-in rules. The simplifier can also use rewrite rules specified in the program. The optimization of a GHC Core program is divided in a series of Core-to-Core translations. The simplifier is one of them. The role of the backend is to generate code for different platforms.

GHC plugins. The plugin mechanism of GHC [6] allows programmers to insert their own passes in the compilation pipeline. We use it to implement a confluence checker. Our plugin receives a Core program, checks local confluence and termination, and outputs the result of the checks but does not modify the Core program.

GHC rewrite rules. The user can use rewrite rules in a Haskell program to teach the compiler optimizing transformations specific to their programs [13]. The syntax of rewrite rules is:

```
{-# RULES
   "name" forall <var>...<var>. f <expr> = <expr>
   ...
 #-}
```

The left-hand side of a rule must be a function application `f <expr>` where the function `f` is in the scope. The left-hand side and right-hand side of the rewrite rules are parsed as Core expressions at the compile time and forms rewrite rules on Core, which we call ***Core rules***.

2.1 Method for checking confluence

To check confluence, we use the well-known Newman's lemma, stating "termination and local confluence implies confluence". To prove local confluence, we should check all possible situations that admit two ways of rewriting, and also to check their convergence. Checking the joinability of critical pairs, we conclude local confluence [2]. Critical pairs can be enumerated by computing overlaps between the left-hand sides of rules using high-order unification.

Example: confluence of map/map. Consider the familiar `map` function on lists, and the following rewrite rule, which translates the composition of two `map`s to a single `map`.

```
{-# RULES
    "map/map" forall f g xs. map f (map g xs) = map (f.g) xs; #-}
```

The rhs is more efficient than the lhs because it traverses a list once rather than twice in the lhs. Therefore, this is an optimisation rule. This is actually an example of "Playing by the Rule" [13]; it is also used in several libraries such as `Data.Text`. Termination of this rule is easy because the number of `map` decreases by each application of the rule. Our tool GSOL outputs the following critical pair. There are two redexes at the topmost expression: the whole expression (rewriting to the lower left) and an inner map expression (rewriting to the lower right).

$$\begin{tikzcd}
& \mathsf{map}(\mathsf{F}, \underline{\mathsf{map}(\mathsf{G}, \mathsf{map}(\mathsf{G}', \mathsf{XS}))}) \arrow[dl, "\text{(map/map)}"'] \arrow[dr, "\text{(map/map)}"] & \\
\mathsf{map}(\mathsf{F} \circ \mathsf{G}, \mathsf{map}(\mathsf{G}', \mathsf{XS})) \arrow[d, "\text{(map/map)}"'] & \text{(MAP)} & \mathsf{map}(\mathsf{F}, \mathsf{map}(\mathsf{G} \circ \mathsf{G}', \mathsf{XS})) \arrow[d, "\text{(map/map)}"] \\
\mathsf{map}((\mathsf{F} \circ \mathsf{G}) \circ \mathsf{G}', \mathsf{XS}) & \neq & \mathsf{map}(\mathsf{F} \circ (\mathsf{G} \circ \mathsf{G}'), \mathsf{XS})
\end{tikzcd}$$

As presented above, this shows a non-joinable critical pair and there are two different normal forms by the $(\mathsf{map}/\mathsf{map})$ rewrite rule. To make the rewrite system confluent, we need to add a rule to make these two equal. One may prefer to add a rule

```
{-# RULES
    "o-assoc" forall f g h.  f.(g.h) = (f.g).h;  #-}
```

By adding this rule, the above critical pair becomes joinable. However, adding this rule induces an additional critical pair, which GSOL points out:

$$\begin{tikzcd}
& \mathsf{F} \circ \underline{(\mathsf{G} \circ (\mathsf{H} \circ \mathsf{H'}))} \arrow[dl, "\text{(o-assoc)}"'] \arrow[dr, "\text{(o-assoc)}"] & \\
(\mathsf{F} \circ \mathsf{G}) \circ (\mathsf{H} \circ \mathsf{H'}) \arrow[d, "\text{(o-assoc)}"'] & \text{(COMP)} & \mathsf{F} \circ ((\mathsf{G} \circ \mathsf{H}) \circ \mathsf{H'}) \arrow[d, "\text{(o-assoc)}"] \\
((\mathsf{F} \circ \mathsf{G}) \circ \mathsf{H}) \circ \mathsf{H'} & = & ((\mathsf{F} \circ \mathsf{G}) \circ \mathsf{H}) \circ \mathsf{H'}
\end{tikzcd}$$

Fortunately, this critical pair is joinable. Moreover, GSOL checks that the system $(\mathsf{map}/\mathsf{map})(\mathsf{o}-\mathsf{assoc})$ is terminating. By applying Newman's lemma, which we infer that the rewrite system $(\mathsf{map}/\mathsf{map})(\mathsf{o}-\mathsf{assoc})$ is confluent.

3 Implementation

SOL. SOL is an implementation of a formal framework of *second-order computation systems*, which is a computational counterpart of second-order algebraic theories [4, 5]. This framework has been used in [9].

Second-order computation systems are based on second-order abstract syntax given by the language of meta-terms [8]:

$$t ::= x \mid x.t \mid f(t_1,\dots,t_n) \mid M[t_1,\dots,t_n].$$

These forms are respectively variables, abstractions, and function terms, and the last form is called a meta-application. A meta-application $M[t_1,\dots,t_n]$ means: when we instantiate M with a term s, free variables of s are replaced with (meta-)terms $t_1,\dots,t_n$. Computation rules are pairs of meta-terms. Based on polymorphic computation systems presented in [10], we modified SOL to cope with higher-order polymorphic computation systems and use it as the checking engine of GSOL.

We implemented GSOL as a Core plugin to check the confluence of GHC rewrite rules. Our plugin is installed into the beginning of the optimization pipeline. The plugin proceeds as the following three steps:

1. Collecting the Core rules.
2. Translating them to SOL rules.
3. Calling the modified SOL for checking. SOL performs the checking functions and print the output to the standard output.

The translation of Core rules is done by applying a structural recursive translation of Core terms to both sides of each rule. It is basically a known encoding method used in [9, 10], which encodes λ-terms to meta-terms. We have developed a stand-alone shell command `gsol` for Haskell files,

4 Example: Arrow

In this section, we demonstrate GSOL by examining the Arrow library of GHC. Arrows [12, 17] provide a way to programming with various computational effects in Haskell. The `Control.Arrow` module in the `base` package of GHC has provided arrows. We excerpt the type class and rules described in the file `Control.Arrow` in Fig. 1.

The `Arrow` inherits the `Category` type class, hence the composition `(.)` is inherited from it. Note that the often used left-to-right arrow composition ">>>" has been defined in terms of `(.)` in

```
module Control.Arrow
class Category a => Arrow a where
  arr    :: (b -> c) -> a b c
  first  :: a b c -> a (b,d) (c,d)
  second :: a b c -> a (d,b) (d,c)
  (***)  :: a b c -> a b' c' -> a (b,b') (c,c')

{-# RULES
"compose/arr"    forall f g .
                 (arr f) . (arr g) = arr (f . g)
"first/arr"      forall f .
                 first (arr f) = arr (first f)
"second/arr"     forall f .
                 second (arr f) = arr (second f)
"product/arr"    forall f g .
                 arr f *** arr g = arr (f *** g)
"compose/first"  forall f g .
                 (first f) . (first g) = first (f . g)
"compose/second" forall f g .
                 (second f) . (second g) = second (f . g)
```

Figure 1: Rules in `Control.Arrow`

`Control.Category`. The method `arr` embeds a pure function into an arrow. The method `first` adds the identity arrow in parallel [11].

Instances of the arrow class must satisfy 9 laws ([17], Fig. 1: Arrow equations), which are based on the semantics of arrows, Freyd categories [11]. The rules in Fig. 1 are some of them. For example, `"compose/arr"` expresses that `arr` is a functor.

Using GSOL, we try to check the confluence of the rules. Let us see the translated polymorphic computation system. It reveals that the rules actually have various type arguments, type class dictionaries and functions (such as `dC`, `$p1Arrow`).

% gsol Arrow.hs -s

```
-- Signature --
 o : (cat:k->k->*,Category(cat),b:k,c:k,a:k,cat(b,c),cat(a,b)) -> cat(a,c)
 arr : (a:*->*->*,Arrow(a),b:*,c:*,(b -> c)) -> a(b,c)
 first : (a:*->*->*,Arrow(a),b:*,c:*,d:*,a(b,c)) -> a(pair(b,d),pair(c,d))
..
 (compose/arr)  o(@cat,dC,@b,@c,@a,arr(@cat,dA,@b,@c,x1.F[x1]),arr(@cat,dA1,@a,@b,x2.G[x2]))
          => arr(@cat,dA,@a,@c,x3.o(%->,$fCategoryTYPE->,@b,@c,@a,F,G,x3))
 (first/arr)  first(@a,dA,@b,@c,@d,arr(@a,dA1,@b,@c,x14.F[x14]))
          => arr(@a,dA,%pair(b,d),%pair(c,d),x15.first(%->,$fArrow->,@b,@c,@d,F,x15))
 (compose/first)  o(@a,dC,%pair(b1,d),%pair(c,d),%pair(b,d),first(@a,dA,@b1,@c,@d,F),first(@a,dA1,@b,@b1,@d,G))
          => first(@a,dA,@b,@c,@d,o(@a,$p1Arrow(@a,dA),@b1,@c,@b,F,G))
```

In `(compose/arr)`, we see that the compositions "o" in lhs and rhs are actually different instances, as the highlighted arguments show. The lhs uses the composition of a category (because of `@cat`), while the rhs uses the composition of pure functions (because of `%->`, meaning the pure function type constructor), although this information was implicit in the source rule. Since `arr` embeds a pure function into an arrow, this description that rhs's composition is for `%->` is correct.

```
(first/arr)   first(@a,dA,@b,@c,@d,arr(@a,dA1,@b,@c,x.F[x]))
         => arr(@a,dA,%pair(b,d),%pair(c,d),x.first(%->,$fArrow->,@b,@c,@d,F,x))
```

Although the source rule `"first/arr"` looks like just swapping `first` and `arr`, the actual polymorphic computation rule `(first/arr)` is not simply so. The lhs' `first` is an instance at a category `a`, while *the rhs' `first` is an instance at the pure function type "`->`".*

Next we try to check local confluence GSOL found that there are 8 critical pairs and all are non-joinable. Let us excerpt one of them with suppressing the type arguments for brevity.

% gsol Arrow.hs cri

```
..
5: Overlap (compose/first)-(first/arr)--- dA'|-> dA, F|-> arr(dA1',F') -------------------------------------------------
   (compose/first) o(dC, first(dA,F) ,first(dA1,G)) => first(dA,o($p1Arrow(dA),F,G))
   (first/arr) first(dA',arr(dA1',F')) => arr(dA',x17'.first($fArrow->,F',x17'))

                                          o(dC,first(dA,arr(dA1',F')),first(dA1,G))
first(dA,o($p1Arrow(dA),arr(dA1',F'),G)) <-(compose/first)-∧-(first/arr)-> o(dC,arr(dA,x.first($fArrow->,F',x)),first(dA1,G))
      ---> first(dA,o($p1Arrow(dA),arr(dA1',F'),G)) =#= o(dC,arr(dA,x.first($fArrow->,F',x)),first(dA1,G)) <---
..
#NON 8 joinable... (Total 8 CPs)
NO
```

The diagram actually expresses the following situation, where dictionaries are attached as subscripts:

$$
\begin{tikzcd}[column sep=small]
& 5 : \underline{\mathsf{first}_{\mathsf{dA}}(\mathsf{arr}_{\mathsf{dA1'}}(\mathsf{F'}))} \bullet_{\mathsf{dC}} \mathsf{first}_{\mathsf{dA1}}(\mathsf{G}) \arrow[dl, "\text{(compose/first)}"'] \arrow[dr, "\text{(first/arr)}"] & \\
\mathsf{first}_{\mathsf{dA}}(\mathsf{arr}_{\mathsf{dA1'}}(\mathsf{F'}) \bullet \mathsf{G}) & \neq & \mathsf{arr}_{\mathsf{dA1'}}(\mathsf{first}_{\$\mathsf{fArrow}\to}(\mathsf{F'})) \bullet_{\mathsf{dC}} \mathsf{first}_{\mathsf{dA1}}(\mathsf{G})
\end{tikzcd}
$$

The entire expression can be rewritten using the rule `(compose/first)` that moves composition ($\bullet$) into the application of `first`. The underlined sub-expression can also be rewritten using the rule `(first/arr)` that swaps the order of application of `first` and `arr`. Since both sides of the critical pair (CP) are normal forms, it shows *non-confluence*.

To make this CP joinable, one way is to add a rule

```
{#- RULES  "not-generally-hold"  forall f.  arr f = f  -#}
```

but it is semantically invalid for general arrows. This law holds only for the arrows of pure functions. Another way is to add a new rule orienting the CP.

In summary, we have found the following by GSOL's confluence checking. Since the type arguments in source rules are implicit, we need to be careful that the rules actually use intended instances. The rewrite rules in `Control.Arrow` are non-confluent, which has not been reported elsewhere.

References

[1] T. Aoto, M. Hamana, N. Hirokawa, A. Middeldorp, J. Nagele, N. Nishida, K. Shintani, and H. Zankl. Confluence Competition 2018. In *Proc. of FSCD 2018*, volume 108 of *LIPIcs*, pages 32:1–32:5, 2018.

[2] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[3] Confluence competition official site, 2020. http://project-coco.uibk.ac.at/2020/.

[4] M. Fiore and C.-K. Hur. Second-order equational logic. In *Proc. of CSL'10*, LNCS 6247, pages 320–335, 2010.

[5] M. Fiore and O. Mahmoud. Second-order algebraic theories. In *Proc. of MFCS'10*, LNCS 6281, pages 368–380, 2010.

[6] Compiler plugins, 2020. https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/extending_ghc.html.

[7] B. Hagedorn, J. Lenfers, et al. Achieving high-performance the functional way: A functional pearl on expressing high-performance optimizations as rewrite strategies. *Proc. ACM Program. Lang.*, 4(ICFP), 2020.

[8] M. Hamana. Free Σ-monoids: A higher-order syntax with metavariables. In *Proc. of APLAS'04*, LNCS 3302, pages 348–363, 2004.

[9] M. Hamana. How to prove decidability of equational theories with second-order computation analyser SOL. *Journal of Functional Programming*, 29(e20), 2019.

[10] M. Hamana, T. Abe, and K. Kikuchi. Polymorphic computation systems: Theory and practice of confluence with call-by-value. *Science of Computer Programming*, 187(102322), 2020.

[11] C. Heunen and B. Jacobs. Arrows, like monads, are monoids. In *Proc. of MFPS 22, ENTCS*, volume 158, pages 219–236, 2006.

[12] J. Hughes. Generalising monads to arrows. *Sci. Comput. Program.*, 37(1-3):67–111, 2000.

[13] S. P. Jones, A. Tolmach, and T. Hoare. Playing by the rules: rewriting as a practical optimisation technique in GHC. In *Haskell Workshop 2001*, 2001.

[14] S. Martin, C. Manuel, S. P. Jones, and D. Kevin. System f with type equality coercions. In *Proc. of TLDI '07*, pages 53–66, 2007.

[15] J. Nagele, B. Felgenhauer, and A. Middeldorp. CSI: New evidence – a progress report. In *Proc. of CADE'17*, LNCS (LNAI) 10395, pages 385–397, 2017.

[16] K. Onozawa, K. Kikuchi, T. Aoto, and Y. Toyama. ACPH: System description. In *6th Confluence Competition (CoCo 2017)*, 2017.

[17] R. Paterson. A new notation for arrows. In *Proceedings of ICFP'07*, pages 229–240, 2001.

Formalized Signature Extension Results for Equivalence*

Alexander Lochmann[1], Fabian Mitterwallner[1], Aart Middeldorp[1]

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

Abstract

Conversion equivalence and normalization equivalence are important properties of two rewrite systems. We investigate how many constants are needed to reduce these properties to their ground versions for linear variable-separated rewrite systems. Our results are implemented in the decision tool FORT-h and formalized in Isabelle/HOL. The latter enables the validation of the proofs produced by the former in the certifier FORTify.

1 Introduction

FORT-s is a tool to synthesize rewrite systems (TRSs) that satisfy a given property expressible in the first-order theory of rewriting. It is based on FORT-h, a tool that implements a decision procedure for the first-order theory of rewriting for the class of linear variable-separated TRSs, which comprises all left-linear right-ground TRSs. We refer to [3,4,6] for further details. It is of interest to synthesize TRSs that depend on one or more other TRSs. This can be done by passing additional TRSs to FORT-s in addition to a formula which references multiple systems. The additional systems are then also passed to the decision procedure. For example, if we want to transform the TRS $\mathcal{R}$ consisting of the rules

$$\mathsf{a} \to \mathsf{b} \qquad\qquad \mathsf{f}(\mathsf{a}) \to \mathsf{b} \qquad\qquad \mathsf{g}(\mathsf{a}, x) \to \mathsf{f}(\mathsf{a})$$

into an equivalent complete (confluent and terminating) TRS, the command

```
fort-s "[0](WCR & SN) & forall s,t ([0]s <->* t <=> [1]s <->* t)" file.trs
```

with `file.trs` containing $\mathcal{R}$ (in COPS format) is executed. The latter is referred to by the index 1 in the formula whereas 0 refers to the TRS to be synthesized. The command returns the TRS $\mathcal{S}$ consisting of the rules

$$\mathsf{a} \to \mathsf{b} \qquad\qquad \mathsf{f}(\mathsf{b}) \to \mathsf{g}(\mathsf{a}, \mathsf{a}) \qquad\qquad \mathsf{g}(\mathsf{b}, \mathsf{b}) \to \mathsf{a}$$

The result is complete (as demanded by `"[0](WCR & SN)"`), but not equivalent to $\mathcal{R}$! The reason is that `"forall s,t ([0]s <->* t <=> [1]s <->* t)"` ensures equivalence on ground terms (since the decision procedure implemented in FORT-h is based on tree automata techniques) but this is not the same as equivalence on all terms; we have $\mathsf{g}(\mathsf{a}, x) \leftrightarrow^*_{\mathcal{R}} \mathsf{a}$ but $\mathsf{g}(\mathsf{a}, x) \leftrightarrow^*_{\mathcal{S}} \mathsf{a}$ does not hold.

In [2] we presented formalized results that allow reducing confluence-related properties (confluence CR, unique normal forms with respect to reduction UNR and conversion UNC, commutation COM) to properties on ground terms by adding fresh constants to the underlying signature. In this note we present similar results for two different notions of equivalence, conversion equivalence and normalization equivalence.

*This work was supported by the Austrian Science Fund (FWF) project P30301.

2 Preliminaries

In this paper we drop the usual constraints on TRSs by allowing terms on the right-hand sides of rules to contain variables not appearing on the left, and left-hand sides to be variables. A rule $\ell \to r$ is called *variable-separated* if $\mathcal{V}\mathsf{ar}(\ell) \cap \mathcal{V}\mathsf{ar}(r) = \varnothing$. A TRS is variable-separated if all its rules are variable-separated. Two TRSs $\mathcal{R}$ and $\mathcal{S}$ over a common signature $\mathcal{F}$ are *conversion equivalent* (CE) if the relations $\leftrightarrow^*_\mathcal{R}$ and $\leftrightarrow^*_\mathcal{S}$ coincide on $\mathcal{T}(\mathcal{F},\mathcal{V})$. They are *ground* conversion equivalent (GCE) if the relations coincide on the set $\mathcal{T}(\mathcal{F})$ of ground terms. We call $\mathcal{R}$ and $\mathcal{S}$ *normalization equivalent* (NE) if the relations $\to^!_\mathcal{R}$ and $\to^!_\mathcal{S}$ coincide on $\mathcal{T}(\mathcal{F},\mathcal{V})$ and *ground* normalization equivalent (GNE) if this holds for $\mathcal{T}(\mathcal{F})$.

A binary predicate P on terms over a given signature $\mathcal{F}$ is closed under *multi-hole contexts* if $P(C[s_1,\dots,s_n],C[t_1,\dots,t_n])$ holds whenever C is a multi-hole context over $\mathcal{F}$ with $n \geqslant 0$ holes and $P(s_i,t_i)$ holds for all $1 \leqslant i \leqslant n$. In the results presented in the next section we make use of the following result from [2]. Here $\to^{*\epsilon*}_\mathcal{A}$ abbreviates $\to^*_\mathcal{A} \cdot \to^\epsilon_\mathcal{A} \cdot \to^*_\mathcal{A}$, so $s \to^{*\epsilon*}_\mathcal{A} t$ if $s \to^*_\mathcal{A} t$ contains a root step.

Lemma 1. *Let $\mathcal{A}$ be a TRS over some signature $\mathcal{F}$ and let P be a binary predicate that is closed under multi-hole contexts over $\mathcal{F}$. If $s \to^{*\epsilon*}_\mathcal{A} t \implies P(s,t)$ for all terms s and t then $s \to^*_\mathcal{A} t \implies P(s,t)$ for all terms s and t.* □

Rewrite sequences involving root steps play an important role for linear variable-separated TRSs since they permit the use of different substitutions for the left and right-hand side of the employed rewrite rule, due to variable separation. We also make use of [2, Lemma 8].

Lemma 2. *Let $\mathcal{R}$ be a linear TRS over a signature $\mathcal{F}$ that contains a constant c which does not appear in $\mathcal{R}$. If $s \to^*_\mathcal{R} t$ with $\mathsf{c} \in \mathcal{F}\mathsf{un}(s) \setminus \mathcal{F}\mathsf{un}(t)$ then $s[u]_p \to^*_\mathcal{R} t$ using the same rewrite rules at the same positions, for all terms u and positions $p \in \mathcal{P}\mathsf{os}(s)$ such that $s|_p = \mathsf{c}$.* □

3 Results

The results in this section are formalized in Isabelle/HOL [1]. Similar to the example in the introduction, the following example shows that the two equivalence properties are not equivalent to their ground versions.

Example 3. The linear variable-separated TRSs

$$\mathcal{R}\colon \quad \mathsf{f}(x) \to \mathsf{a} \qquad\qquad \mathcal{S}\colon \quad \mathsf{f}(\mathsf{a}) \to \mathsf{a} \qquad \mathsf{f}(\mathsf{f}(\mathsf{a})) \to \mathsf{a}$$

over the signature $\mathcal{F} = \{\mathsf{f},\mathsf{a}\}$ are neither normalization equivalent nor conversion equivalent as can be seen from $\mathsf{f}(x) \to^!_\mathcal{R} \mathsf{a}$ and $\mathsf{f}(x) \not\leftrightarrow^*_\mathcal{S} \mathsf{a}$. Since every ground term rewrites in $\mathcal{R}$ and $\mathcal{S}$ to the unique ground normal form a, the TRSs are ground normalization equivalent as well as ground conversion equivalent. However, adding a single fresh constant c to the signature is sufficient to reproduce the counterexample: $\mathsf{f}(\mathsf{c}) \to^!_\mathcal{R} \mathsf{a}$ and $\mathsf{f}(\mathsf{c}) \not\leftrightarrow^*_\mathcal{S} \mathsf{a}$. So the TRSs are neither ground normalization equivalent nor ground conversion equivalent over the extended signature $\mathcal{F} \uplus \{\mathsf{c}\}$.

In later proofs we will limit the rewrite sequences under consideration to those containing root steps by instantiating Lemma 1

- with $P_1(s,t)\colon s \to^*_{\mathcal{S}\cup\mathcal{S}^-} t$ and $\mathcal{R} \cup \mathcal{R}^-$ for $\mathcal{A}$ in proofs related to CE, and
- with $P_2(s,t)\colon t \in \mathsf{NF}_\mathcal{R} \implies s \to^*_\mathcal{S} t$ and $\mathcal{R}$ for $\mathcal{A}$ in proofs related to NE.

Note the identity $\to_{\mathcal{R}\cup\mathcal{R}^-} = \leftrightarrow_{\mathcal{R}}$ in the first case. We also use the symmetric instances, with $\mathcal{R}$ and $\mathcal{S}$ switching places, for both properties. Both P_1 and P_2 are closed under multi-hole contexts. By considering only sequences containing root steps we can use different substitutions on the left and right of the sequence, due to variable-separation. These substitutions will usually introduce fresh constants in the terms. We will also use Lemma 2 in subsequent proofs to remove these additional constants from rewrite sequences as follows. Let σ_c denote the substitution mapping all variables to c. If $s\sigma_c \to_{\mathcal{R}}^* t$ then $s \to_{\mathcal{R}}^* t$ by repeatedly applying Lemma 2 (to each occurrence of c in $s\sigma_c$), assuming c appears neither in $\mathcal{R}$ nor in t.

A single fresh constant suffices to reduce conversion equivalence to ground conversion equivalence.

Theorem 4. *Linear variable-separated TRSs $\mathcal{R}$ and $\mathcal{S}$ over a common signature $\mathcal{F}$ such that $\mathcal{T}(\mathcal{F}) \neq \varnothing$ are conversion equivalent if and only if $\mathcal{R}$ and $\mathcal{S}$ are ground conversion equivalent over the signature $\mathcal{F} \uplus \{c\}$.*

Proof. For the if direction we assume that $\mathcal{R}$ and $\mathcal{S}$ are ground conversion equivalent over $\mathcal{F} \uplus \{c\}$. Due to Lemma 1 (instantiated with P_1) and symmetry, it suffices to show the inclusion $\leftrightarrow_{\mathcal{R}}^{*\epsilon*} \subseteq \leftrightarrow_{\mathcal{S}}^*$ on terms in $\mathcal{T}(\mathcal{F},\mathcal{V})$. Suppose $s \leftrightarrow_{\mathcal{R}}^{*\epsilon*} t$. Let $d \in \mathcal{F}$ be a constant, whose existence is guaranteed by the assumption $\mathcal{T}(\mathcal{F}) \neq \varnothing$, and consider the substitutions σ_c and σ_d mapping all variables to the constants c and d respectively. Closure under substitutions and variable separation yields $s\sigma_c \leftrightarrow_{\mathcal{R}}^{*\epsilon*} t\sigma_c$ and $s\sigma_c \leftrightarrow_{\mathcal{R}}^{*\epsilon*} t\sigma_d$. Ground conversion equivalence gives $s\sigma_c \leftrightarrow_{\mathcal{S}}^* t\sigma_c$ and $s\sigma_c \leftrightarrow_{\mathcal{S}}^* t\sigma_d$, and thus also $t\sigma_c \leftrightarrow_{\mathcal{S}}^* t\sigma_d$. Using Lemma 2 yields $s \leftrightarrow_{\mathcal{S}}^* t\sigma_d$ and $t \leftrightarrow_{\mathcal{S}}^* t\sigma_d$. Hence $s \leftrightarrow_{\mathcal{S}}^* t$ as desired.

For the only-if direction we assume that $\mathcal{R}$ and $\mathcal{S}$ are conversion equivalent over $\mathcal{F}$. Consider $s \leftrightarrow_{\mathcal{R}}^* t$ with $s,t \in \mathcal{T}(\mathcal{F} \uplus \{c\})$ and let $\phi_x^c(\cdot)$ be the function that replaces all occurrences of the constant c with the variable x in terms. Since the constant c does not appear in $\mathcal{R}$, we obtain $\phi_x^c(s) \leftrightarrow_{\mathcal{R}}^* \phi_x^c(t)$ from $s \leftrightarrow_{\mathcal{R}}^* t$. Conversion equivalence yields $\phi_x^c(s) \leftrightarrow_{\mathcal{S}}^* \phi_x^c(t)$. By choosing a variable $x \notin \mathcal{V}\mathsf{ar}(s) \cup \mathcal{V}\mathsf{ar}(t)$, the latter implies $s \leftrightarrow_{\mathcal{S}}^* t$ by closure under substitutions. □

Two fresh constants are required to reduce normalization equivalence to its ground version.

Theorem 5. *Linear variable-separated TRSs $\mathcal{R}$ and $\mathcal{S}$ over a common signature $\mathcal{F}$ are normalization equivalent if and only if $\mathcal{R}$ and $\mathcal{S}$ are ground normalization equivalent over $\mathcal{F} \uplus \{c,d\}$.*

Proof. The only-if direction can be proved with the methods used in the proof of Theorem 4. For the if direction we assume that $\mathcal{R}$ and $\mathcal{S}$ are ground normalization equivalent over $\mathcal{F} \uplus \{c,d\}$, which implies $\mathsf{NF}_{\mathcal{R}} = \mathsf{NF}_{\mathcal{S}}$. Hence, it remains to show that $s \to_{\mathcal{R}}^{*\epsilon*} t$ with $t \in \mathsf{NF}_{\mathcal{R}}$ implies $s \to_{\mathcal{S}}^* t$ due to Lemma 1 (instantiated with P_2) and symmetry. From $s \to_{\mathcal{R}}^{*\epsilon*} t$ we obtain $s\sigma_c \to_{\mathcal{R}}^* t\sigma_d$, as the involved root step allows independent substitutions on the left and right-hand sides. Moreover, $t\sigma_d \in \mathsf{NF}_{\mathcal{R}}$, since d does not occur in $\mathcal{R}$. From ground normalization equivalence we obtain $s\sigma_c \to_{\mathcal{S}}^* t\sigma_d$. Finally, Lemma 2 allows the removal of the substitutions, resulting in the desired rewrite sequence $s \to_{\mathcal{S}}^* t$. □

Contrary to Theorem 4 one fresh constant is not sufficient as shown in the following example.

Example 6. Consider the two linear variable-separated TRSs

$$\begin{array}{llll}
\mathcal{R}\colon & \mathsf{a} \to \mathsf{b} & \mathsf{f}(\mathsf{f}(x,y),z) \to \mathsf{f}(\mathsf{b},\mathsf{b}) & \mathsf{f}(\mathsf{b},x) \to \mathsf{f}(\mathsf{b},\mathsf{b}) \\
& \mathsf{f}(x,\mathsf{a}) \to \mathsf{f}(z,\mathsf{b}) & & \\
\mathcal{S}\colon & \mathsf{a} \to \mathsf{b} & \mathsf{f}(\mathsf{f}(x,y),z) \to \mathsf{f}(\mathsf{b},\mathsf{b}) & \mathsf{f}(\mathsf{b},x) \to \mathsf{f}(\mathsf{b},\mathsf{b}) \\
& \mathsf{f}(\mathsf{b},\mathsf{a}) \to \mathsf{f}(z,\mathsf{b}) & \mathsf{f}(\mathsf{f}(x,y),\mathsf{a}) \to \mathsf{f}(z,\mathsf{b}) &
\end{array}$$

They are not normalization equivalent since $\mathsf{f}(x,\mathsf{a}) \to^!_{\mathcal{R}} \mathsf{f}(z,\mathsf{b})$ and $\mathsf{f}(x,\mathsf{a}) \not\to^*_{\mathcal{S}} \mathsf{f}(z,\mathsf{b})$. The TRSs are however ground normalization equivalent over the signature $\mathcal{F} \uplus \{\mathsf{c}\}$. First observe that the only ground normal forms reachable via a rewrite sequence involving a root step are b and $\mathsf{f}(\mathsf{c},\mathsf{b})$. The normal form b is reached (using a root step) only from a, in both $\mathcal{R}$ and $\mathcal{S}$. The normal form $\mathsf{f}(\mathsf{c},\mathsf{b})$ can be reached from all ground terms of the shape $\mathsf{f}(t,\mathsf{a})$. For $\mathcal{R}$ this is obvious and for $\mathcal{S}$ this can be seen by a case analysis on the root symbol of t. Adding a second constant d allows one to mimick the original counterexample since $\mathsf{f}(\mathsf{c},\mathsf{a}) \to^!_{\mathcal{R}} \mathsf{f}(\mathsf{d},\mathsf{b})$ and $\mathsf{f}(\mathsf{c},\mathsf{a}) \not\to^*_{\mathcal{S}} \mathsf{f}(\mathsf{d},\mathsf{b})$.

For left-linear right-ground TRSs, a single fresh constant is enough to reduce normalization equivalence to ground normalization equivalence.

Theorem 7. *Left-linear right-ground TRSs $\mathcal{R}$ and $\mathcal{S}$ over a common signature $\mathcal{F}$ are normalization equivalent if and only if $\mathcal{R}$ and $\mathcal{S}$ are ground normalization equivalent over $\mathcal{F} \uplus \{c\}$.*

Proof. We mention the differences with the proof of Theorem 5. For the identity of $\mathsf{NF}_{\mathcal{R}}$ and $\mathsf{NF}_{\mathcal{S}}$ for arbitrary terms, a single constant suffices. If $s \to^{*\epsilon*}_{\mathcal{R}} t$ then t is ground. Hence $s\sigma_c \to^*_{\mathcal{R}} t$ and thus $s\sigma_c \to^*_{\mathcal{S}} t$ by ground normalization equivalence. Lemma 2 gives $s \to^*_{\mathcal{S}} t$. □

Each additional constant increases the execution time of $\mathsf{FORT\text{-}h}$ significantly. Hence results that reduce the required number are of obvious interest. For example, ground TRSs need no additional constants for the properties described in this paper. In the remainder of this section we present results for TRSs over *monadic* signatures, which are signatures that consist of constants and unary function symbols. In [6, Lemma 6] it is shown that for left-linear right-ground TRSs and properties related to confluence, no additional constants are needed. The same holds for commutation, which is a new (and formalized) result.

Theorem 8. *Right-ground TRSs $\mathcal{R}$ and $\mathcal{S}$ over a common monadic signature $\mathcal{F}$ commute if and only if $\mathcal{R}$ and $\mathcal{S}$ ground commute.*

Proof. The only-if direction trivially holds. For the if direction we assume that $\mathcal{R}$ and $\mathcal{S}$ ground commute. Consider $s \to^*_{\mathcal{R}} t$ and $s \to^*_{\mathcal{S}} u$ for $s,t,u \in \mathcal{T}(\mathcal{F},\mathcal{V})$. If $s = t$ or $s = u$, then $t \to^*_{\mathcal{S}} \cdot {}_{\mathcal{R}}^*\!\!\leftarrow u$ obviously holds. So suppose $s \to^+_{\mathcal{R}} t$ and $s \to^+_{\mathcal{S}} u$. Since $\mathcal{F}$ is monadic and $\mathcal{R}$ and $\mathcal{S}$ are right-ground, we infer that t and u are ground terms. Let $r \in \mathcal{T}(\mathcal{F})$ be an arbitrary ground term and let σ_r be the substitution which replaces all variables in $\mathcal{V}\mathsf{ar}(s)$ by r. Since $t\sigma_r = t$, $u\sigma_r = u$ and $\to^+$ is closed under substitution, we obtain $s\sigma_r \to^+_{\mathcal{R}} t$ and $s\sigma_r \to^+_{\mathcal{S}} u$. Ground commutation yields the desired joining sequence $t \to^*_{\mathcal{S}} \cdot {}_{\mathcal{R}}^*\!\!\leftarrow u$. □

Note that Theorem 8 cannot be extended to linear variable-separated TRSs which require two constants even for monadic signatures, as seen by the TRS $\{\mathsf{a} \to x\}$. It does not commute with itself, since $\mathsf{a} \to x$ and $\mathsf{a} \to y$, but it ground commutes with itself over the signatures $\{\mathsf{a}\}$ and $\{\mathsf{a},\mathsf{c}\}$.

Unlike commutation, the properties NE and CE require additional constants for TRSs over monadic signatures even for left-linear right-ground systems, as can be seen from Example 3. Nevertheless, we can reduce the number of constants to one if the signature is monadic, even if the restriction to left-linear right-ground TRSs is dropped. A key observation is that in non-empty rewrite sequences in a linear variable-separated TRS over a monadic signature fresh constants can be replaced by arbitrary terms.

Lemma 9. *Let $\mathcal{R}$ be a variable-separated TRS over a monadic signature $\mathcal{F}$ that contains a constant c which does not appear in $\mathcal{R}$. If $s \to^+_{\mathcal{R}} t$ and $p \in \mathcal{P}\mathsf{os}(s)$ such that $s|_p = c$ then $s[u]_p \to^+_{\mathcal{R}} t$ using the same rewrite rules at the same positions, for all terms u.* □

Table 1: Additional constants required to reduce a property P to ground P.

property	left-linear right-ground TRSs		linear variable-separated TRSs		
CE	$\underline{1}$	($\underline{1}$)	$\underline{1}$	($\underline{1}$)	(Theorem 4)
NE	$\underline{1}$	($\underline{1}$)	$\underline{2}$	($\underline{1}$)	(Theorems 5, 7, 10)
COM	$1^{\dagger}$	($\underline{0}$)	$2^{\dagger}$	($\underline{2}$)	(Theorem 8)
CR	$1^{*\dagger}$	$(0)^{*}$	$2^{\dagger}$	($\underline{2}$)	
SCR	1^{*}	$(0)^{*}$	$\underline{2}$	($\underline{2}$)	
WCR	1^{*}	$(0)^{*}$	$\underline{2}$	($\underline{2}$)	
UNR	$1^{*\dagger}$	$(0)^{*}$	$2^{\dagger}$	($\underline{2}$)	
UNC	$2^{*\dagger}$	$(0)^{*}$	$2^{\dagger}$	($\underline{2}$)	
NFP	1^{*}	$(0)^{*}$	$\underline{2}$	($\underline{2}$)	

As variable-separated TRSs are closed under inverse we can immediately deduce that rewrite sequences of the shape $s\sigma_c \to_{\mathcal{R}}^{+} t\sigma_c$ imply $s \to_{\mathcal{R}}^{+} t$ for monadic systems. With this we are ready to prove our claim.

Theorem 10. *Variable-separated TRSs $\mathcal{R}$ and $\mathcal{S}$ over a common monadic signature $\mathcal{F}$ are normalization equivalent if and only if $\mathcal{R}$ and $\mathcal{S}$ are ground normalization equivalent over $\mathcal{F} \uplus \{c\}$.*

Proof. Note that TRSs over a monadic signature are necessarily linear. We mention the differences with the proof of Theorem 5. A single constant suffices to prove $\mathsf{NF}_{\mathcal{R}} = \mathsf{NF}_{\mathcal{S}}$. Consider a rewrite sequence $s \to_{\mathcal{R}}^{*\epsilon*} t$ with $t \in \mathsf{NF}_{\mathcal{R}}$. Ground normalization equivalence and substitution closure yields $s\sigma_c \to_{\mathcal{S}}^{!} t\sigma_c$. Furthermore, since the sequence $s \to_{\mathcal{R}}^{*\epsilon*} t$ is non-empty by definition, $s\sigma_c \notin \mathsf{NF}_{\mathcal{R}} = \mathsf{NF}_{\mathcal{S}}$ and thus $s\sigma_c \neq t\sigma_c$ as $t\sigma_c \in \mathsf{NF}_{\mathcal{S}}$. Hence $s\sigma_c \to_{\mathcal{S}}^{+} t\sigma_c$. Applying Lemma 9 twice allows us to replace all occurrences of c in $s\sigma_c$ and $t\sigma_c$ by the corresponding variables, resulting in $s \to_{\mathcal{S}}^{*} t$. $\square$

In Table 1 we summarize the results of this paper as well as the related results (the final six rows) from [2]. The numbers for TRSs over monadic signatures are given in parentheses. The underlined numbers are new results. The results marked with an asterisk are proved in [5], those marked with a dagger are formalized in [2].

4 Conclusion

In this paper we presented new signature extension results allowing us to reduce the problem of proving CE and NE to GCE and GNE respectively for linear variable-separated TRSs (Theorems 4 and 5). This is done by adding fresh constants to the signature. We also showed that the number of required fresh constants for reducing NE to GNE can be reduced for left-linear right-ground TRSs as well as for monadic systems (Theorem 10). The latter was also shown for the property COM (Theorem 8). All results are formalized in Isabelle/HOL [1] and implemented in the tools FORT-h, FORT-s, and the certifier FORTify. Binaries of the tools can be obtained from

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

The implemented results enable FORT-s to find an equivalent complete TRS of our leading example using the formula

```
"[0](WCR & SN) & {+1} forall s,t ([0]s <->* t <=> [1]s <->* t)"
```

The {+1} instructs the decision procedure to add one fresh constant to the signature when evaluating the subformula for CE. Calling FORT-s with this formula on our leading example $\mathcal{R}$ produces the TRS:

$$\mathsf{a} \to \mathsf{b} \qquad \mathsf{f}(\mathsf{b}) \to \mathsf{g}(\mathsf{a},\mathsf{a}) \qquad \mathsf{g}(\mathsf{a},x) \to \mathsf{a}$$

which is indeed complete and equivalent to $\mathcal{R}$ on all terms (not just ground terms). For ease of use we also added the shorthands CE and NE to the formula language. When using these the tools FORT-h, FORT-s and FORTify add the appropriate amount of constants for any given input TRS.

References

[1] Alexander Lochmann. Reducing rewrite properties to properties on ground terms. *Archive of Formal Proofs*, 2022. https://isa-afp.org/entries/Rewrite_Properties_Reduction.html, Formal proof development.

[2] Alexander Lochmann, Fabian Mitterwallner, and Aart Middeldorp. Formalized signature extension results for confluence, commutation and unique normal forms. In *Proc. 10th International Workshop on Confluence*, pages 25–30, 2021.

[3] Fabian Mitterwallner, Alexander Lochmann, Aart Middeldorp, and Bertram Felgenhauer. Certifying proofs in the first-order theory of rewriting. In *Proc. 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 12652 of *Lecture Notes in Computer Science*, pages 127–144, 2021. doi:10.1007/978-3-030-72013-1_7.

[4] Franziska Rapp and Aart Middeldorp. Automating the first-order theory of left-linear right-ground term rewrite systems. In *Proc. 1st FSCD*, volume 52 of *Leibniz International Proceedings in Informatics*, pages 36:1–36:12, 2016. doi:10.4230/LIPIcs.FSCD.2016.36.

[5] Franziska Rapp and Aart Middeldorp. Confluence properties on open terms in the first-order theory of rewriting. In *Proc. 5th International Workshop on Confluence*, pages 26–30, 2016.

[6] Franziska Rapp and Aart Middeldorp. FORT 2.0. In *Proc. 9th International Joint Conference on Automated Reasoning*, volume 10900 of *Lecture Notes in Artificial Intelligence*, pages 81–88, 2018. doi:10.1007/978-3-319-94205-6_6.

Proving Confluence with CONFident (short version)*

Raúl Gutiérrez[2], Miguel Vítores[1], and Salvador Lucas[1]

[1] DSIC, Universitat Politècnica de València, Spain
mvitvic@posgrado.upv.es, slucas@dsic.upv.es
[2] DLSIIS, Universidad Politécnica de Madrid, Spain
r.gutierrez@upm.es

Abstract

This paper describes the proof framework used in CONFident, a framework to deal with different types of systems (term rewriting systems, context-sensitive term rewriting systems and conditional term rewriting systems) and different types of tasks (checking joinability of critical pairs, termination of systems, feasibility of reachibility problems or deducibility) and different techniques for proving confluence (including modular decompositions, transformations, etc.).

1 Introduction

CONFident is a tool for automatically proving confluence of *Term Rewriting Systems* (TRSs), *Context-Sensitive Term Rewriting Systems* (CS-TRSs), and *Conditional Term Rewriting Systems* (CTRSs) based on a unified framework that combines different processors that can use external calls to prove or disprove confluence of a target system. In general, because of the *undecidability* of confluence, it is quite difficult to find a proof of confluence based on the application of one technique only. On the contrary, a proof of confluence usually involves different phases where the input system is simplified, decomposed, analyzed to extract good properties (linearity, orthogonality,...), transformed into a (hopefully) simpler system or into a critical pair joinability problem.

For this purpose, we have developed a *Confluence Framework*, inspired by the *Dependency Pair Framework*, originally developed for proving (innermost) termination of TRSs [3, 4]. In the Confluence Framework, confluence is treated as a *confluence problem* which is transformed, decomposed, simplified, etc., into a set of problems by using the so-called *processors*. Besides confluence problems, we also handle *joinability problems* which are produced by some processors acting on confluence problem, and are also treated by appropriate processors. Processors can then be applied again and again on the obtained problems. They often require calls to external tools to solve proof obligations like *termination*, *feasibility*, *theorem proving*, etc. The obtained proof is depicted as a *proof tree* from which confluence of the targetted rewrite system can be proved or disproved. Of course, due to undecidability of confluence such a proof tree may *fail* to be completed in some cases.

In the following, we describe how we manage to combine all those different tasks into a unified framework.

*Partially supported by grant RTI2018-094403-B-C32 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe", and PROMETEO/2019/098.

2 Confluence Framework

In the confluence framework we describe two types of problems that appear naturally in proofs of confluence: confluence problems, that allow us to encapsulates the different variants of input rewrite systems (in our case, TRSs, CS-TRSs or CTRSs); and the joinability problems, which describe a joinability problem generated by a (conditional) critical pair.

A replacement map μ is a mapping from function symbols $f \in \mathcal{R}$ to set of positions $\mu(f) \subseteq \{1, \ldots, k\}$ (k is the arity of f) indicating the *active arguments* where reductions are allowed [8]. We use $\hookrightarrow$ to define the rewriting relation induced by $\mathcal{R}$ and μ.

Definition 1 (Confluence Problems)**.** *We consider two different types of problems:*

- *Let $\mathcal{R}$ be a CTRS (TRSs are included). A* confluence problem*, denoted $CR(\mathcal{R})$, is* positive *if $\mathcal{R}$ is confluent; otherwise it is* negative*.*
- *Let $\mathcal{R}$ be a TRS and μ be a replacement map. A μ-confluence problem, denoted $CR(\mathcal{R}, \mu)$, is* positive *if $\mathcal{R}$ is μ-confluent (confluent using $\hookrightarrow$); otherwise it is* negative*.*

A *conditional pair* is $\pi = \langle u, v\rangle \Leftarrow c$ where c is a condition that contains a possibly empty sequence of atomic conditions of the form $s \to^* t$, $s \downarrow t$, $s \leftrightarrow^* t$, and also one-step μ-rewriting problems $s \hookrightarrow t$. If c is empty we just write $\langle u, v\rangle$. We say that π is joinable if for all substitutions σ such that $\sigma(c)$ holds.

Definition 2 (Joinability Problems)**.** *We consider two different types of problems:*

- *Let $\mathcal{R}$ be a CTRS and π be a* conditional pair *where the conditional part has no occurrence of $\hookrightarrow$. A* joinability problem*, denoted $JO(\mathcal{R}, \pi)$, is* positive *if π is joinable in $\mathcal{R}$; otherwise it is* negative*.*
- *Let $\mathcal{R}$ be a TRS, μ be a replacement map, and π be a* conditional pair *where the conditional part contains at most an occurrence of $\hookrightarrow$. A μ-*joinability problem*, denoted $JO(\mathcal{R}, \mu, \pi)$, is* positive *if π is μ-joinable in $\mathcal{R}$; otherwise it is* negative*.*

A processor P accepts a problem τ and returns a set $\mathsf{P}(\tau)$ of (possibly heterogeneous) problems $\tau_1, \ldots, \tau_n$. Those problems can be recasted as proofs (or refutations) of other *problems.* The resulting problems are, hopefully, easier to prove.

Definition 3. *A* processor P *is a partial function from problems into sets of problems; alternatively it can return "*no*". The domain of* P *(i.e., the set of problems for which* P *returns) is denoted $\mathcal{D}om(\mathsf{P})$. We say that* P *is*

- sound *if for all $\tau \in \mathcal{D}om(\mathsf{P})$, τ is positive whenever $\mathsf{P}(\tau) \neq$ "no" and all $\tau' \in \mathsf{P}(\tau)$ are positive.*
- complete *if for all $\tau \in \mathcal{D}om(\mathsf{P})$, τ is negative whenever $\mathsf{P}(\tau) =$ "no" or τ' is negative for some $\tau' \in \mathsf{P}(\tau)$.*

Confluence problems can be proved positive or negative in the usual way by using a proof tree generated by the application of different processors.

2.1 List of processors

CONFident implements several processors. In this section, we give a short description and appropriate reference to them.

Simplification (P_{Simp})**.** Before attempting a proof of confluence of a system $\mathcal{R}$, some simplifications are often possible: (1) *Removing or transforming rules (CS-TRSs, and CTRSs)* of the form $t \to t$ or $t \to t \Leftarrow c$ for some term t, (2) *Removing infeasible rules (CTRSs only)* using infChecker [6], and (3) *Inlining conditional rules (CTRSs only)* as explained in [14, Def. 4].

$$\mathsf{P}_{Simp}(CR(\mathcal{R})) = \{CR(\mathcal{R}')\}$$

Modular decomposition (P_{MD})**.** For TRSs $\mathcal{R}$, processor P_{MD} tries to find a *decomposition* of $\mathcal{R}$ into two TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ if some of the *modularity conditions for confluence* is achieved: (1) disjoint TRSs [16]; or (2) constructor-sharing and left-linear TRSs [13]; or (3) constructor-sharing and layer-preserving TRSs [11].

$$\mathsf{P}_{MD}(CR(\mathcal{R})) = \{CR(\mathcal{R}_1), CR(\mathcal{R}_2)\}$$

Orthogonality (P_{HL})**.** Processor P_{HL} implements: (1) for TRSs $\mathcal{R}$, Huet-Levy's Theorem (*weakly orthogonal TRSs are confluent*, see [12, Sect. 4.3]); (2) for CS-TRSs $(\mathcal{R}, \mu)$, the result in [9, Coro. 36] (*left-orthogonal CS-TRSs with no extended μ-critical pairs are confluent*[1]) [9, Def. 29]; (3) for 3-CTRSs[2] the result in [12, Theorem 7.4.14][3] (*level-confluence* implies confluence); and also (4) the result in [12, Theorem 7.4.11] (orthogonal, properly oriented and right-stable CTRSs are confluent).

$$\mathsf{P}_{HL}(CR(\mathcal{R})) = \emptyset \qquad \mathsf{P}_{HL}(CR(\mathcal{R}, \mu)) = \emptyset$$

Extended Huet processor (P_{Huet})**.** P_{Huet} checks, (1) for CTRSs, joinability of (conditional) critical pairs $\pi_1, \ldots, \pi_n$, using [12, Def. 7.1.8]; (2) for CS-TRSs $(\mathcal{R}, \mu)$, joinability of extended μ-critical pairs $\pi_1, \ldots, \pi_n$. For both uses,P_{Huet} is *not* sound (joinability of critical pairs, alone, does not imply confluence), although it is *complete* ([12, Lemma 4.2.3] for TRSs and [9, Theorem 30] for CS-TRSs).

$$\mathsf{P}_{Huet}(CR(\mathcal{R})) = \{JO(\mathcal{R}, \pi_1), \ldots, JO(\mathcal{R}, \pi_n)\} \quad \mathsf{P}_{Huet}(CR(\mathcal{R}, \mu)) = \{JO(\mathcal{R}, \mu, \pi_1), \ldots, JO(\mathcal{R}, \mu, \pi_n)\}$$

Extended Huet-Newman processor (P_{HN})**.** Relying on termination and μ-termination, P_{HN} applies (1) for TRSs $\mathcal{R}$ with critical pairs $\pi_1, \ldots, \pi_n$ if $\mathcal{R}$ is terminating [1, Corollary 6.2.6]; (2) For CS-TRSs $(\mathcal{R}, \mu)$, with extended μ-critical pairs $\pi_1, \ldots, \pi_n$ if $\mathcal{R}$ is μ-terminating [9, Theorem 32]; or (3) For *normal* CTRSs[4], iff $\mathcal{R}$ is left-linear, terminating, and $\pi_1, \ldots, \pi_n$ are joinable overlays [2, Def. 8]

$$\mathsf{P}_{HN}(CR(\mathcal{R})) = \{JO(\mathcal{R}, \pi_1), \ldots, JO(\mathcal{R}, \pi_n)\} \quad \mathsf{P}_{HN}(CR(\mathcal{R}, \mu)) = \{JO(\mathcal{R}, \mu, \pi_1), \ldots, JO(\mathcal{R}, \mu, \pi_n))\}$$

Confluence as canonical joinability of μ-critical pairs (P_{CanJ})**.** By relying on [5, Thm. 2], we can use a context-sensitive transformations to try to prove confluence of a TRS. In this case, if $\mathcal{R}$ is a left-linear and level-decreasing TRS [5, Def. 1 & 2] (see also [8, Sect. 8.5]), $\mu = \mu^{can}_{\mathcal{R}}$[5], $\pi_1, \ldots, \pi_n$ are the μ-critical pairs of $\mathcal{R}$ [8, Def. 8.5], and $\mathcal{R}$ is μ-terminating.

$$\mathsf{P}_{CanJ}(CR(\mathcal{R})) = \{JO(\mathcal{R}, \mu, \pi_1), \ldots, JO(\mathcal{R}, \mu, \pi_n)\}$$

[1] Such CS-TRSs are called μ-orthogonal [9, Def. 35].

[2] CTRSs of type 3 (usually called 3-CTRSs) are those whose rules $\ell \to r \Leftarrow c$ satisfy that variables occurring in r occur in ℓ or c [10].

[3] Originally in [15, Theorem 4.6]

[4] A CTRS is normal if for all rules $\ell \to r \Leftarrow c$ and conditions $s \approx t$ in c, term t is an irreducible ground term [2, Def. 2].

[5] The *canonical replacement map* $\mu^{can}_{\mathcal{R}}$ for a TRS $\mathcal{R}$ is *the most restrictive replacement map ensuring that the non-variable subterms of the left-hand sides of the rules of $\mathcal{R}$ are all active* [8, Section 5]

Confluence as canonical μ-confluence (P_{CanCR}). By relying on [8, Coro. 8.23], if $\mathcal{R}$ is left-linear and normalizing TRS (i.e., every term has a normal form), and $\mu = \mu^{can}_{\mathcal{R}}$, we can also obtain a *sound* but *not* complete transformation.

$$\mathsf{P}_{CanCR}(CR(\mathcal{R})) = \{CR(\mathcal{R}, \mu)\}$$

Confluence of CTRS as confluence of TRSs ($\mathsf{P}_{\mathcal{U}}$). Processor $\mathsf{P}_{\mathcal{U}}$ transforms a confluence problem for a deterministic and terminating 3-CTRS $\mathcal{R}$ into a confluence problem for a TRS $\mathcal{U}(\mathcal{R})$, where $\mathcal{U}$ is the transformation from [12, Def. 7.2.33] 3-CTRSs to TRSs described in [12, Def. 7.2.48]. This processor is *sound* but *not* complete.

$$\mathsf{P}_{\mathcal{U}}(CR(\mathcal{R})) = \{CR(\mathcal{U}(\mathcal{R}))\}$$

Joinability processor (P_{JO}). P_{JO} implements methods for proving and disproving joinability of conditional pairs are described in [7, Section 6].

$$\mathsf{P}_{JO}(JO(\mathcal{R}, \pi)) = \begin{cases} \emptyset & \text{if } \pi \text{ is joinable w.r.t. } \mathcal{R} \\ \mathsf{no} & \text{if } \pi \text{ is not joinable w.r.t. } \mathcal{R} \end{cases}$$

$$\mathsf{P}_{JO}(JO(\mathcal{R}, \mu, \pi)) = \begin{cases} \emptyset & \text{if } \pi \text{ is } \mu\text{-joinable w.r.t. } \mathcal{R} \\ \mathsf{no} & \text{if } \pi \text{ is not } \mu\text{-joinable w.r.t. } \mathcal{R} \end{cases}$$

2.2 Strategy

Given a rewrite system $\mathcal{R}$, the processors enumerated in Section 2.1 are used to build a proof tree with root $CR(\mathcal{R})$ or $CR(\mathcal{R}, \mu)$ to hopefully conclude (μ-)confluence or non-(μ-)confluence of $\mathcal{R}$. In our proof strategy, we use two strategy combinators, the *sequential* combinator and the *alternative* combinator. The strategies used in an alternative combinator can be executed in parallel. The proof strategy used in CONFident experimentation is as follows: (i) it tries to apply P_{Simp} to simplify the input system or its rules; then (ii) it tries to decompose the problem applying P_{MD}; (iii) at this point, there is a triple alternative with the identity processors (that returns the same input system), P_{CanCR} and $\mathsf{P}_{\mathcal{U}}$; (iv) for each branch, there is an alternative of P_{HL}, P_{HN}, P_{Huet} and P_{CanCR}; and (v) it tries P_{JO} on each joinability problem. Returning "yes", "no" or "maybe" depends on the result of the evaluation of the resulting proof tree.

2.3 Application interface and experimental results

CONFident is written in Haskell and it has more than 80 Haskell files with more than 9000 lines of pure code (blanks and comments not included). The tool is accessible online through its web interface in `http://zenon.dsic.upv.es/confident/`.

CONFident participated in the 2021 International Confluence Competition (CoCo)[6] in the categories TRS, SRS, and CTRS, obtaining the first place in the CTRS category. With respect to context-sensitive rewriting, a subcategory of confluence of CSR will be part of CoCo 2022 (see `http://project-coco.uibk.ac.at/2022/`).

[6] `http://project-coco.uibk.ac.at/2021/`

2.4 Conclusions

CONFident is a tool which is able to automatically prove and disprove confluence of variants of rewrite systems: TRSs, CS-TRSs, and Join, Oriented, and Semi-Equational CTRSs. The proofs are obtained by combining different techniques in what we call *Confluence Framework*, where *confluence* and *joinability* problems are handled (simplified, transformed, etc.) by means of *processors*, which can be freely combined to obtain the proofs which are displayed as a *proof tree*. To the best of our knowledge, CONFident is the only tool which is able to prove and disprove confluence of Join and Semi-Equational CTRSs, and the only tool which is able to prove and disprove confluence of CS-TRSs. CONFident has proved to be a powerful tool for proving confluence of CTRSs. This is witnessed by the first position obtained in the CTRS subcategory of CoCo 2021.

References

[1] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[2] Nachum Dershowitz, Mitsuhiro Okada, and G. Sivakumar. Confluence of Conditional Rewrite Systems. In Stéphane Kaplan and Jean-Pierre Jouannaud, editors, *Conditional Term Rewriting Systems, 1st International Workshop, Orsay, France, July 8-10, 1987, Proceedings*, volume 308 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 1987.

[3] Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Montevideo, Uruguay, March 14-18, 2005, Proceedings*, volume 3452 of *Lecture Notes in Computer Science*, pages 301–331. Springer, 2004.

[4] Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Mechanizing and improving dependency pairs. *J. Autom. Reasoning*, 37(3):155–203, 2006.

[5] Bernhard Gramlich and Salvador Lucas. Generalizing Newman's Lemma for Left-Linear Rewrite Systems. 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 66–80. Springer, 2006.

[6] 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, Paris, France, July 1-4, 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 416–435. Springer, 2020.

[7] Raúl Gutiérrez, Salvador Lucas, and Miguel Vítores. Confluence of Conditional Rewriting in Logic Form. In M. Bojańczy and C. Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*, volume 213 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[8] Salvador Lucas. Context-sensitive Rewriting. *ACM Comput. Surv.*, 53(4):78:1–78:36, 2020.

[9] Salvador Lucas, Miguel Vítores, and Raúl Gutiérrez. Proving and disproving confluence of context-sensitive rewriting. *Journal of Logical and Algebraic Methods in Programming*, 126:100749, 2022.

[10] Aart Middeldorp and Erik Hamoen. Completeness results for basic narrowing. *Appl. Algebra Eng. Commun. Comput.*, 5:213–253, 1994.

[11] Enno Ohlebusch. On the modularity of confluence of constructor-sharing term rewriting systems. In Sophie Tison, editor, *Trees in Algebra and Programming - CAAP'94, 19th International Colloquium, Edinburgh, UK, April 11-13, 1994, Proceedings*, volume 787 of *Lecture Notes in Computer Science*, pages 261–275. Springer, 1994.

[12] Enno Ohlebusch. *Advanced Topics in Term Rewriting.* Springer, 2002.

[13] Jean-Claude Raoult and Jean Vuillemin. Operational and semantic equivalence between recursive programs. *J. ACM*, 27(4):772–796, 1980.

[14] Thomas Sternagel. *Reliable Confluence Analysis of Conditional Term Rewrite Systems.* PhD thesis, Faculty of Mathematics, Computer Science and Physics, University of Innsbruck, August 2017.

[15] Taro Suzuki, Aart Middeldorp, and Tetsuo Ida. Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In Jieh Hsiang, editor, *Rewriting Techniques and Applications, 6th International Conference, RTA-95, Kaiserslautern, Germany, April 5-7, 1995, Proceedings*, volume 914 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 1995.

[16] Yoshihito Toyama. On the church-rosser property for the direct sum of term rewriting systems. *J. ACM*, 34(1):128–143, 1987.

Confluence Competition 2022

Rául Gutiérrez[1], Aart Middeldorp[2], Naoki Nishida[3], and Kiraku Shintani[4]

[1] Universidad Politécnica de Madrid, Madrid, Spain
[2] Department of Computer Science, University of Innsbruck, Austria
[3] Department of Computing and Software Systems, Nagoya University, Japan
[4] School of Information Science, JAIST, Japan

The next few pages in these proceedings contain the descriptions of the tools participating in the 11th Confluence Competition (CoCo 2022). 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 13 tools (listed in order of registration) participating in 11 categories (listed in order of first appearance in CoCo):

	TRS	CPF-TRS	CTRS	GCR	UNR	UNC	NFP	COM	INF	SRS	CSR
CSI	✓	✓			✓	✓	✓			✓	
FORT-h		✓		✓	✓	✓	✓	✓			
FORTify		✓		✓	✓	✓	✓	✓			
CONFident	✓		✓							✓	✓
infChecker									✓		
Toma									✓		
Hakusan	✓									✓	
CoLL								✓			
CeTA		✓									
CO3			✓						✓		
ACP	✓	✓	✓			✓		✓		✓	
AGCP				✓							
NaTT									✓		

The winning (for combined YES/NO answers) tools[1] of CoCo 2021 participated as demonstration tools, to provide a benchmark to measure progress. The live run of CoCo 2022 on StarExec [2] can be viewed at `http://cocograph.uibk.ac.at/2022.html`. Further information about CoCo 2022, including a description of the categories and detailed results, can be obtained from

`http://project-coco.uibk.ac.at/2022/`

Acknowledgements The CoCo steering committee is grateful to Nao Hirokawa and Fabian Mitterwallner for their support.

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.

[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.

[1]They are not listed in the table but see `http://project-coco.uibk.ac.at/2021/results.php`.

CoCo 2022 Participant: CSI 1.2.6

Fabian Mitterwallner and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria
fabian.mitterwallner@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. 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 [5]. Some of the implemented techniques are described in [1,4,7]. CSI can also produce certificates for confluence results, which are checked by CeTA. Compared to last year's version, CSI 1.2.6 can now produce certificates containing proofs based on development-closed critical pairs, which is a sufficient condition for confluence of left-linear TRSs [6]. These can be checked by the latest version of CeTA [3], due to the formalization and certification efforts by Christina Kohl, parts of which are described in [2].

CSI participates in the following CoCo 2022 categories: CPF-TRS, NFP, SRS, TRS, UNC, and UNR.

References

[1] Bertram Felgenhauer. Confluence for Term Rewriting: Theory and Automation. PhD thesis, University of Innsbruck, 2015.

[2] Christina Kohl and Aart Middeldorp. Development Closed Critical Pairs: Towards a Formalized Proof. In *Proc. 11th International Workshop on Confluence*, 2022. This volume.

[3] Christina Kohl, René Thiemann, and Aart Middeldorp. CoCo 2022 Participant: CeTA 2.42. In *Proc. 11th International Workshop on Confluence*, 2022. This volume.

[4] Julian Nagele. Mechanizing Confluence: Automated and Certified Analysis of First- and Higher-Order Rewrite Systems. PhD thesis, University of Innsbruck, 2017.

[5] 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.

[6] Vincent van Oostrom. Developing Developments. *Theoretical Computer Science*, 175(1):159–181, 1997. doi: 10.1016/S0304-3975(96)00173-9.

[7] Harald Zankl. Challenges in Automation of Rewriting. Habilitation thesis, University of Innsbruck, 2014.

CoCo 2022 Participant: FORT-h 2.0*

Fabian Mitterwallner and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria
fabian.mitterwallner@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 [1]. FORT-h implements a new variant, described in [2], of the decision procedure 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 assistent Isabelle/HOL.

Compared to last year's version, FORT-h 2.0 makes use of improved signature extension results for proving properties on non-ground terms [5]. More specifically, FORT-h is now able add fewer fresh constants to the signature depending on certain syntactic properties of the input systems. This shrinks the size of the constructed automata leading to significantly faster execution times in some cases. A command-line version of FORT-h 2.0 can be downloaded from

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

FORT-h participates in the following CoCo 2022 categories: COM, GCR, NFP, UNC, and UNR. Together with FORTify [6], it participates in CPF-TRS in addition to the previously mentioned ones to produce certified YES/NO answers.

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.

[2] Fabian Mitterwallner, Alexander Lochmann, Aart Middeldorp, and Bertram Felgenhauer. Certifying Proofs in the First-Order Theory of Rewriting. In *Proc. 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 12652 of *LNCS*, pages 127–144, 2021. doi: 10.1007/978-3-030-72013-1_7.

[3] Franziska Rapp and Alexander Middeldorp. Automating the First-Order Theory of Left-Linear Right-Ground Term Rewrite Systems. In *Proc. 1st International Conference on Formal Structures for Computation and Deduction*, volume 52 of *Leibniz International Proceedings in Informatics*, pages 36:1–36:12, 2016. doi: 10.4230/LIPIcs.FSCD.2016.36.

[4] 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.

[5] Alexander Lochmann, Fabian Mitterwallner, and Aart Middeldorp. Formalized Signature Extension Results for Equivalence In *Proc. 11th International Workshop on Confluence*, 2022. This volume.

[6] Alexander Lochmann, Fabian Mitterwallner, and Aart Middeldorp. CoCo 2022 Participant: FORTify 2.0. In *Proc. 11th International Workshop on Confluence*, 2022. This volume.

*Supported by FWF (Austrian Science Fund) project P30301.

CoCo 2022 Participant: FORTify 2.0*

Alexander Lochmann, Fabian Mitterwallner, and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria

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 recently been formalized [2,3] in Isabelle/HOL. The code generation facilities of Isabelle then give rise to the certifier FORTify [4] 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)/

Compared to last year, the formalization on which FORTify is based contains more and improved signature extension results, as described in [5]. Importantly for this competition, it contains new results related to the *normal form property* making it possible for FORTify to compete in the NFP category. Furthermore, it adds fewer constants for ground rewrite systems and for rewrite systems contain only unary function symbols and constants (monadic systems). This leads to smaller automata in the procedure and in turn to shorter run times in some cases. Other performance improvements are also included.

This year FORTify participates, together with FORT-h, in the following CoCo 2022 categories: CPF-TRS, COM, GCR, NFP, UNC, and UNR.

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.

[2] Alexander Lochmann and Aart Middeldorp. Formalized Proofs of the Infinity and Normal Form Predicates in the First-Order Theory of Rewriting. In *Proc. 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 12079 of *LNCS*, pages 178–194, 2020. doi: 10.1007/978-3-030-72013-1_7.

[3] Alexander Lochmann, Aart Middeldorp, Fabian Mitterwallner, and Bertram Felgenhauer. Formalizing the First-Order Theory of Rewriting. In *Proc. 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 250–263, 2021. doi: 10.1145/3437992.3439918.

[4] Fabian Mitterwallner, Alexander Lochmann, Aart Middeldorp, and Bertram Felgenhauer. Certifying Proofs in the First-Order Theory of Rewriting. In *Proc. 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 12652 of *LNCS*, pages 127–144, 2021. doi: 10.1007/978-3-030-72013-1_7.

[5] Alexander Lochmann, Fabian Mitterwallner, and Aart Middeldorp. Formalized Signature Extension Results for Equivalence In *Proc. 11th International Workshop on Confluence*, 2022. This volume.

[6] Fabian Mitterwallner and Aart Middeldorp. CoCo 2022 Participant: FORT-h 2.0. In *Proc. 11th International Workshop on Confluence*, 2022. This volume.

*Supported by FWF (Austrian Science Fund) project P30301.

infChecker at the 2022 Confluence Competition*

Raúl Gutiérrez[1], Salvador Lucas[2], and Miguel Vítores[2]

[1] Universidad Politécnica de Madrid, Madrid, Spain
r.gutierrez@upm.es
[2] VRAIN, Universitat Politècnica de València, Valencia, Spain
slucas@dsic.upv.es
mvitvic@posgrado.upv.es

1 Overview

infChecker is a tool for checking *(in)feasibility* of goals $\mathcal{G} = \{F_i\}_{i=1}^m$ where $F_i = (s_{ij} \bowtie_{ij} t_{ij})_{i=1}^{n_i}$ and $\bowtie_{ij} \in \{\to, \to^*, \to^+, \hookrightarrow, \hookrightarrow^*, \hookrightarrow^+, \rhd, \unrhd, \downarrow, \Downarrow, \leftrightarrow, \leftrightarrow, \leftrightarrow^*, \leftrightarrow^*\}$ where predicates $\bowtie_{ij}$ represent binary relations on terms (most of them well-known or easy generalizations of well-known relations) defined by provability of goals $s \bowtie_{ij} t$ with respect to a *first-order theories* $\mathsf{Th}_{\bowtie_{ij}}$ [1, 3].

The tool is available here:

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

In 2022, we participate using the version presented at the 2021 Confluence Competition. A short description of the tool can be seen in [2].

infChecker participated from 2019 until today in the confluence competition (CoCo) in the INF category. Currently, infChecker is the most powerful tool for proving and disproving infeasibility.

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, S.Lucas and M. Vítores. infChecker at the 2021 Confluence Competition. In: Mimram S., Rocha C. (eds) 10th International Workshop on Confluence, IWC 2021. 2021.

[3] S. Lucas and R. Gutiérrez. Use of Logical Models for Proving Infeasibility in Term Rewriting. *Information Processing Letters*, 136:90–95, 2018.

*Partially supported by the EU (FEDER) and the Spanish MCIU under grant RTI2018-094403-B-C32 and by the Spanish Generalitat Valenciana under grant PROMETEO/2019/098.

Toma 0.2: An Equational Theorem Prover

Teppei Saito and Nao Hirokawa

JAIST, Japan

Toma is an automatic theorem prover for first-order equational systems, freely available at:

https://www.jaist.ac.jp/project/maxcomp/

The typical usage is: `toma --inf <file>`, where `<file>` is an infeasibility problem in the CoCo format [5]. The tool outputs `YES` if infeasibility of the problem is shown, and `MAYBE` otherwise. It also accepts the TPTP CNF format [6].

Toma proves infeasibility as follows: By using the *split-if* encoding [2] a given infeasibility problem is transformed into a word problem of form $\mathcal{E} \vdash \mathsf{T} \not\approx \mathsf{F}$ whose validity entails infeasibility of the original problem. The word problem is solved by a new variant of maximal (ordered) completion [7, 3]:

1. Given an equational system $\mathcal{E}_1$, we construct a lexicographic path order $\succ_{\mathsf{lpo}}$ that maximizes reducibility of the ordered rewrite system $(\mathcal{E}_1, \succ_{\mathsf{lpo}})$ [7].
2. Using the order, we run ordered completion [1] on $\mathcal{E}_1$. Here we do not employ the deduce rule (critical pair generation). Such a run eventually ends with an inter-reduced version $(\mathcal{E}_2, \succ_{\mathsf{lpo}})$ of $(\mathcal{E}_1, \succ_{\mathsf{lpo}})$.
3. The tool checks ground-completeness of the ordered rewrite system $(\mathcal{E}_2, \succ_{\mathsf{lpo}})$ by Martin and Nipkow's method [4].
   (a) If $(\mathcal{E}_2, \succ_{\mathsf{lpo}})$ is ground-complete but T and F are not joinable, the tool outputs `YES` and terminates.
   (b) If T and F are joinable in $(\mathcal{E}_2, \succ_{\mathsf{lpo}})$, the tool outputs `MAYBE` and terminates.
   (c) Otherwise, there exists at least one equation that is valid in $\mathcal{E}_2$ but not ground-joinable in $(\mathcal{E}_2, \succ_{\mathsf{lpo}})$. Let $\mathcal{E}_3$ be a set of such equations. Setting $\mathcal{E}_1 := \mathcal{E}_2 \cup \mathcal{E}_3$, the tool goes back to the first step.

References

[1] L. Bachmair, N. Dershowitz and D. A. Plaisted. Completion without Failure. *Resolution of Equations in Algebraic Structures vol. 2: Rewriting* , pp. 1–30, Academic Press, 1989.

[2] K. Claessen and N. Smallbone. Efficient Encodings of First-Order Horn Formulas in Equational Logic. *Proc. 9th IJCAR*, LNCS 10900, pp. 388–404, 2018.

[3] N. Hirokawa. Completion and Reduction Orders. *Proc. 6th FSCD*, LIPIcs, vol. 195, pp. 2:1-2:9, 2021.

[4] U. Martin and T. Nipkow. Ordered Rewriting and Confluence. *Proc. 10th CADE*, LNCS 499, pp. 366–380, 1990.

[5] A. Middeldorp, J. Nagele, and K. Shintani. Confluence Competition 2019. *Proc. 25th TACAS*, LNCS 11429, pp. 25–40, 2019.

[6] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning, vol. 59*, no. 4, pp. 483–502, 2017.

[7] S. Winkler and G. Moser. Mædmax: A Maximal Ordered Completion Tool. *Proc. 9th IJCAR*, LNCS 10900, pp. 472–480, 2018.

Hakusan 0.5: A Confluence Tool

Kiraku Shintani and Nao Hirokawa

JAIST, Japan

`s1820017@jaist.ac.jp`, `hirokawa@jaist.ac.jp`

Hakusan is a prototype tool for automatically proving confluence of left-linear term rewrite systems (TRSs). The tool, written in Haskell, is freely available at:

http://www.jaist.ac.jp/project/saigawa/

The typical usage is: `hakusan <file>`. Here the input file is written in the TRS format [3]. The tool outputs `YES` if confluence of the input TRS is proved, and `MAYBE` if the tool does not reach any conclusion. Currently the tool does not support non-confluence analysis.

Confluence analysis in Hakusan is based on *compositional* confluence criteria [4], which mean sufficient conditions such that, given a rewrite system $\mathcal{R}$ and its subsystem $\mathcal{C} \subseteq \mathcal{R}$, confluence of $\mathcal{C}$ implies that of $\mathcal{R}$. Compositional criteria can be seen as a combination method for confluence analysis. Hakusan alternately uses two compositional confluence criteria: One is a compositional version of the rule labeling method [6, Theorem 56], and the other is a compositional version of the confluence criterion by critical pair systems [1].

Theorem 1. *Let $\mathcal{R}$ be a left-linear TRS and $\mathcal{C}$ a confluent TRS with $\mathcal{C} \subseteq \mathcal{R}$, and also let ϕ and ψ be labeling functions from $\mathcal{R}$ to $\mathbb{N}$. The TRS $\mathcal{R}$ is confluent if we have $\mathcal{R}_{\phi,0} = \mathcal{C} = \mathcal{R}_{\psi,0}$ and the following conditions hold for all $(k,m) \in \mathbb{N}^2 \setminus \{(0,0)\}$.*

- *Every parallel critical peak of form $t \mathrel{{}_{\phi,k}{\leftarrow\!\!\!\!\!\!\shortmid\!\shortmid}} s \xrightarrow{\epsilon}_{\psi,m} u$ is (ψ,ϕ)-decreasing.*
- *Every parallel critical peak of form $t \mathrel{{}_{\psi,m}{\leftarrow\!\!\!\!\!\!\shortmid\!\shortmid}} s \xrightarrow{\epsilon}_{\phi,k} u$ is (ϕ,ψ)-decreasing.*

Here $\mathcal{R}_{\phi,k}$ stands for $\{\ell \to r \in \mathcal{R} \mid \phi(\ell \to r) \leqslant k\}$ and $\mathrel{\shortmid\!\shortmid\!\!\!\!\!\!\rightarrow}_{\phi,k}$ for the parallel step of $\mathcal{R}_{\phi,k}$. See [4, Definition 27] for the definition of (ψ,ϕ)-decreasingness.

Theorem 2. *Let $\mathcal{R}$ be a left-linear TRS and $\mathcal{C}$ a confluent TRS with $\mathcal{C} \subseteq \mathcal{R}$. The TRS $\mathcal{R}$ is confluent if $\mathrel{{}_{\mathcal{R}}{\leftarrow\!\!\!\!\!\!\shortmid\!\shortmid}} \rtimes \xrightarrow{\epsilon}_{\mathcal{R}} \subseteq \to^*_{\mathcal{R}} \cdot {}^*_{\mathcal{R}}\!\leftarrow$ and $\mathcal{P}/\mathcal{R}$ is terminating. Here $\mathcal{P}$ stands for the TRS: $\{s \to t, s \to u \mid t \mathrel{{}_{\mathcal{R}}{\leftarrow\!\!\!\!\!\!\shortmid\!\shortmid}} s \xrightarrow{\epsilon}_{\mathcal{R}} u$ is a parallel critical peak but not $t \leftrightarrow^*_{\mathcal{C}} u\}$.*

For automation, the tool employs the SMT solver Z3 [2] for finding suitable labeling functions, and the termination tool NaTT [5] for testing relative termination.

References

[1] N. Hirokawa and A. Middeldorp. Decreasing diagrams and relative termination. Journal of Automated Reasoning, volume 47, pages 481–501, 2011.

[2] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Proc. 12th TACAS*, volume 4963 of LNCS, pages 337–340, 2008.

[3] A. Middeldorp, J. Nagele, and K. Shintani. Confluence Competition 2019. *Proc. 25th TACAS*, volume 11429 of *LNCS*, pages 25–40, 2019.

[4] K. Shintani and N. Hirokawa. Compositional Confluence Criteria. In *Proc. 7th FSCD*, 2022.

[5] A. Yamada and K. Kusakari and T. Sakabe. Nagoya Termination Tool. In *Proc. 25th RTA*, volume 8560 of *LNCS*, pages 446–475, 2014.

[6] H. Zankl, B. Felgenhauer, and A. Middeldorp. Labelings for decreasing diagrams. Journal of Automated Reasoning, volume 54, pages 101–133, 2015.

CoLL 1.6.1: A Commutation Tool

Kiraku Shintani

JAIST, Japan
`s1820017@jaist.ac.jp`

`CoLL` (version 1.6.1) is a tool for automatically proving commutation of left-linear term rewrite systems (TRSs). The tool, written in OCaml, is freely available at:

http://www.jaist.ac.jp/project/saigawa/coll/

The typical usage is: `coll <file>`. Here the input file is written in the commutation problem format [4]. The tool outputs `YES` if commutation of the input TRSs is proved, `NO` if non-commutation is shown, and `MAYBE` if the tool does not reach any conclusion.

In this tool commutation of left-linear TRSs is shown by *Hindley's Commutation Theorem*:

Theorem 1 ([2, 7])**.** *ARSs* $\mathcal{A} = \langle A, \{\to_\alpha\}_{\alpha \in I}\rangle$ *and* $\mathcal{B} = \langle A, \{\to_\beta\}_{\beta \in J}\rangle$ *commute if* $\to_\alpha$ *and* $\to_\beta$ *commute for all* $\alpha \in I$ *and* $\beta \in J$*.*

Here indexes are interpreted as subsystems of the input TRSs. For every pair of subsystems the tool proves the commutation property, employing the three criteria: simultaneous closedness [5], parallel closedness [9], parallel upside closedness and outside closedness [6], rule labeling with weight function [10, 1], and Church-Rosser modulo A/C [3]. A detailed description of `CoLL` can be found in [8].

References

[1] T. Aoto. Automated confluence proof by decreasing diagrams based on rule-labelling. In *Proc. 21st RTA*, volume 6 of *LIPIcs*, pages 7–16, 2010.

[2] J. R. Hindley. *The Church-Rosser Property and a Result in Combinatory Logic*. PhD thesis, University of Newcastle-upon-Tyne, 1964.

[3] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4):1155–1194, 1986.

[4] A. Middeldorp, J. Nagele, and K. Shintani. Confluence Competition 2019. *Proc. 25th TACAS*, volume 11429 of *LNCS*, pages 25–40, 2019.

[5] S. Okui. Simultaneous critical pairs and Church–Rosser property. In *Proc. 9th RTA*, volume 1379 of *LNCS*, pages 2–16, 1998.

[6] M. Oyamaguchi and Y. Ohta. On the open problems concerning Church-Rosser of left-linear term rewriting systems. *IEICE Transactions on Information and Systems*, 87(2):290–298, 2004.

[7] B. K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20:160–187, 1973.

[8] K. Shintani and N. Hirokawa. CoLL: A confluence tool for left-linear term rewrite systems. In *Proc. 25th CADE*, volume 9195 of *LNAI*, pages 127–136, 2015.

[9] Y. Toyama. On the Church-Rosser property of term rewriting systems. NTT ECL Technical Report, No.17672, NTT, 1981.

[10] V. van Oostrom. Confluence by decreasing diagrams converted. In A. Voronkov, editor, *Proc. 19th RTA*, volume 5117 of *LNCS*, pages 306–320, 2008.

CoCo 2022 Participant: CeTA 2.42

Christina Kohl, René Thiemann, and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria

The tool CeTA [4] 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. 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/

In the following, we describe what is new in version 2.42 of CeTA.

The most important extension with respect to CoCo is the ability to check confluence of left-linear TRSs via van Oostrom's development closedness criterion [5]. It is trivial to use this criterion in a certificate, since the application conditions are checked automatically without requiring further information on the joins. Some important steps of the formalization effort are described in [1]. Note that, although every parallel closed TRS is also development closed, this latter method in CeTA 2.42 does not (yet) completely subsume the parallel closedness criterion that was added in version 2.25 [3]. The reason is that the formalization of parallel closedness allows a weakened joining condition for overlays—an extension known as *almost* parallel closed. As shown in [5] the same extension can be applied to the development closedness criterion but its formalization in IsaFoR is currently still work in progress.

Finally the efficiency of the parser inside CeTA has been improved for version 2.42 by providing a custom encoding of characters. More details on this can be found in [2].

References

[1] Christina Kohl and Aart Middeldorp. Development Closed Critical Pairs: Towards a Formalized Proof. In *Proc. 11th International Workshop on Confluence*, 2022. This volume.

[2] Christina Kohl and René Thiemann. CeTA– A Certifier for termCOMP 2022. In *Proc. 18th International Workshop on Termination*, 2022. To appear.

[3] Julian Nagele and Aart Middeldorp. Certification of Classical Confluence results for Left-Linear Term Rewrite Systems. In *Proc. 7th International Conference on Interactive Theorem Proving*, volume 9807 of *Lecture Notes in Computer Science*, pages 290–306, 2016. doi: 10.1007/978-3-319-43144-4_18.

[4] René Thiemann and Christian Sternagel. Certification of Termination Proofs using CeTA. In *Proc. 22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468, 2009. doi: 10.1007/978-3-642-03359-9_31.

[5] Vincent van Oostrom. Developing Developments. *Theoretical Computer Science*, 175(1):159–181, 1997. doi: 10.1016/S0304-3975(96)00173-9.

CO3 (Version 2.3)

Naoki Nishida and Misaki Kojima

Nagoya University, Nagoya, Japan
{nishida@, k-misaki@trs.css.}i.nagoya-u.ac.jp

CO3, a converter for proving confluence of conditional TRSs,[1] tries to prove confluence of conditional term rewrite systems (CTRSs, for short) by using a transformational approach (cf. [7]). 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* $\mathbb{U}$ [9], 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 [8, 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 prove 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.

The current version accepts both *join* and *semi-equational* CTRSs, and transforms them into equivalent DCTRSs to prove confluence or infeasibility [6].

This version has two improvements compared with the previous one [6]. One is the removal of valid conditions: For a conditional rule $\ell \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k \in \mathcal{R}$, a *valid* conditions $s_i \twoheadrightarrow t_i$ such that either s_i is a variable appearing once in the rule or $s_i \to t_i$ is in $\mathcal{R}$ as an unconditional rule is removed from the conditional part. The other is the computation of approximated edges for estimated dependency graphs: For dependency pairs $s \to t, u \to v$ of a TRS $\mathcal{R}$, if t is ground and $\mathit{REN}(\mathit{CAP}(t))$ is not unifiable with u, then we check whether there exists a reduct t' of t such that $\mathit{REN}(\mathit{CAP}(t'))$ is unifiable with u (cf. a *narrowing processor* [1]).

References

[1] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *J. Autom. Reason.*, 37(3):155–203, 2006.

[2] K. Gmeiner, B. Gramlich, and F. Schernhammer. On soundness conditions for unraveling deterministic conditional rewrite systems. In *Proc. RTA 2012*, vol. 15 of *LIPIcs*, pp. 193–208, 2012.

[3] K. Gmeiner, N. Nishida, and B. Gramlich. Proving confluence of conditional term rewriting systems via unravelings. In *Proc. IWC 2013*, pp. 35–39, 2013.

[4] Y. Maeda, N. Nishida, M. Sakai, and T. Kobayashi. Extending narrowing trees to basic narrowing in term rewriting. IEICE Tech. Rep. SS2018-39, Vol. 118, No. 385, pp. 73–78, 2019, in Japanese.

[5] N. Nishida. CO3 (Version 2.1). In *Proc. IWC 2020*, page 67, 2020.

[6] N. Nishida. CO3 (Version 2.2). In *Proc. IWC 2021*, page 61, 2021.

[7] N. Nishida, T. Kuroda, and K. Gmeiner. CO3 (Version 1.3). In *Proc. IWC 2016*, p. 74, 2016.

[8] N. Nishida and Y. Maeda. Narrowing trees for syntactically deterministic conditional term rewriting systems. In *Proc. FSCD 2018*, vol. 108 of *LIPIcs*, pp. 26:1–26:20, 2018.

[9] E. Ohlebusch. Termination of logic programs: Transformational methods revisited. *Appl. Algebra Eng. Commun. Comput.*, 12(1/2):73–116, 2001.

[1] http://www.trs.css.i.nagoya-u.ac.jp/co3/

ACP: System Description for CoCo 2022

Takahito Aoto

Institute of Science and Technology, Niigata University
aoto@ie.niigata-u.ac.jp

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) and the commutation property of term rewriting systems. The ingredients of the former property have been appeared in [2, 5]. Our (dis)proofs of commutation are based on a development closed criterion [6] and a simple search for counter examples. We are now working toward extending our confluence prover for CTRSs [4]. No new criterion, however, has been incorporated from the one submitted for CoCo 2021.

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 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] R. Haga, Y. Kagaya and T. Aoto. A critical pair criterion for level-commutation of conditional term rewriting systems. In *Proc. of 11th IWC*, this volume.

[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.

AGCP: System Description for CoCo 2022

Takahito Aoto

Institute of Science and Technology, Niigata University
aoto@ie.niigata-u.ac.jp

AGCP (Automated Groud Confluence Prover) [1] is a tool for proving ground confluence of many-sorted term rewriting systems. AGCP is written in Standard ML of New Jersey (SML/NJ). AGCP proves ground confluence of many-sorted term rewriting systems based on two ingredients. One ingredient is to divide the ground confluence problem of a many-sorted term rewriting system $\mathcal{R}$ into that of $\mathcal{S} \subseteq \mathcal{R}$ and the inductive validity problem of equations $u \approx v$ w.r.t. $\mathcal{S}$ for each $u \to r \in \mathcal{R} \setminus \mathcal{S}$. Here, an equation $u \approx v$ is inductively valid w.r.t. $\mathcal{S}$ if all its ground instances $u\sigma \approx v\sigma$ is valid w.r.t. $\mathcal{S}$, i.e. $u\sigma \overset{*}{\leftrightarrow}_{\mathcal{S}} v\sigma$. Another ingredient is to prove ground confluence of a many-sorted term rewriting system via the *bounded ground convertibility* of the critical pairs. Here, an equation $u \approx v$ is said to be bounded ground convertibile w.r.t. a quasi-order $\succsim$ if $u\theta_g \xleftrightarrow[\succsim]{*}_{\mathcal{R}} v\theta_g$ for any its ground instance $u\sigma_g \approx v\sigma_g$, where $x \xleftrightarrow[\succsim]{*} y$ iff there exists $x = x_0 \leftrightarrow \cdots \leftrightarrow x_n = y$ such that $x \succsim x_i$ or $y \succsim x_i$ for every x_i.

Rewriting induction [3] is a well-known method for proving inductive validity of many-sorted term rewriting systems. In [1], an extension of rewriting induction to prove bounded ground convertibility of the equations has been reported. Namely, for a reduction quasi-order $\succsim$ and a quasi-reducible many-sorted term rewriting system $\mathcal{R}$ such that $\mathcal{R} \subseteq \succ$, the extension proves bounded ground convertibility of the input equations w.r.t. $\succsim$. The extension not only allows to deal with non-orientable equations but also with many-sorted TRSs having non-free constructors. Several methods that add wider flexibility to the this approach are given in [2]: when suitable rules are not presented in the input system, additional rewrite rules are constructed that supplement or replace existing rules in order to obtain a set of rules that is adequate for applying rewriting induction; and an extension of the system of [2] is used if if the input system contains non-orientable constructor rules. AGCP uses these extension of the rewriting induction to prove not only inductive validity of equations but also the bounded ground convertibility of the critical pairs. Finally, some methods to deal with disproving ground confluence are added as reported in [2].

No new ground (non-)confluence criterion has been incorporated from the one submitted for CoCo 2021.

References

[1] T. Aoto and Y. Toyama. Ground confluence prover based on rewriting induction. In *Proc. of 1st FSCD*, volume 52 of *LIPIcs*, pages 33:1–33:12. Schloss Dagstuhl, 2016.

[2] T. Aoto, Y. Toyama and Y. Kimura. Improving Rewriting Induction Approach for Proving Ground Confluence. In *Proc. of 2nd FSCD*, volume 84 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl, 2017.

[3] U.S. Reddy. Term rewriting induction. In *Proc. of CADE-10*, volume 449 of *LNAI*, pages 162–177. Springer-Verlag, 1990.

NaTT in CoCo 2022

Akihisa Yamada

National Institute of Advanced Industrial Science and Technology, Japan
akihisa.yamada@aist.go.jp

NaTT [3] is a termination prover for plain term rewriting, written in OCaml and the source code is available at:

https://www.trs.cm.is.nagoya-u.ac.jp/NaTT/

Though NaTT is not a confluence prover, since the last year it is participating in the infeasibility category of the Confluence Competition, as its quick reachability checker [1] can be used to solve infeasibility problems.

In this year, NaTT solves infeasibility problems by generalizing its term ordering techniques, the core of termination proving. The details of the technique will be presented at IJCAR 2022 [2].

Since the last year, the input format of NaTT is a (relatively simple) XML, now described at the above page. In the competition, it reads the COPS format by translating it into the XML format, using the text-to-and-from-XML translator TXtruct, which was presented at WPTE 2022. The format description in the above page is automatically generated by TXtruct.

References

[1] Christian Sternagel and Akihisa Yamada. Reachability analysis for termination and confluence of rewriting. In Tomás Vojnar and Lijun Zhang, editors, *TACAS (1) 2019*, volume 11427 of *LNCS*, pages 262–278. Springer, 2019. doi:10.1007/978-3-030-17462-0_15.

[2] Akihisa Yamada. Term orderings for non-reachability of (conditional) rewriting. In *IJCAR 2022*, volume 13385 of *LNCS*, 2022. To appear.

[3] Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. Nagoya Termination Tool. In Gilles Dowek, editor, *RTA-TLCA 2014*, volume 8560 of *LNCS*, pages 466–475. Springer, 2014. doi:10.1007/978-3-319-08918-8_32.