Proceedings of the

15th International Workshop on Confluence

July 24, 2026

Lisbon, Portugal

Foreword

The 15th International Workshop on Confluence (IWC 2026) is held on July 24, 2026, in Lisbon, Portugal, co-located with the 11th International Conference on Formal Structures for Computation and Deduction (FSCD 2026) and the International Joint Conference on Automated Reasoning (IJCAR 2026), as part of the Federate Logic Conference (FLoC).

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 2026 continues this tradition. The present report comprises ten regular submissions, and the abstract of the invited talk by Jean-Pierre Jouannaud, as well as descriptions of tools participating in the 15th Confluence Competition (CoCo 2026). The contributions in these proceedings illustrate the diversity of current research on confluence. They cover topics ranging from critical pair analysis, completion techniques, reachability analysis and confluence criteria for various rewriting formalisms to the formalization of confluence results and implementation aspects, highlighting the continued importance of confluence across a broad spectrum of rewriting formalisms and computational models.

The maturity in confluence research in the last decade resulted in a variety of novel approaches, which were also implemented in powerful tools that compete in the annual confluence competition. The second part of this report devoted to CoCo 2026 provides a general overview as well as system descriptions of all competition entrants.

IWC 2026 was made possible by the commitment of many people who contributed to the submissions, the preparation and the program of the workshop, as well as the confluence competition. These include authors of papers and tools, committee members, external reviewers, and the organizers of CoCo, as well as the local organizers. Their hard work is very much appreciated.

Raúl Gutiérrez and René Thiemann — Valencia and Innsbruck, 24 July 2026

Organization

IWC Steering Committee

- Naoki Nishida, Nagoya University, Japan
- Sarah Winkler, Free University of Bozen-Bolzano, Italy

IWC Program Committee

- Takahito Aoto, Niigata University, Japan
- Thiago Felicissimo, INRIA, France
- Carsten Fuhs, Birkbeck, University of London, UK
- Raúl Gutiérrez, Universitat Politècnica de València, Spain (co-chair)
- Ievgen Ivanov, Taras Shevchenko National University of Kyiv, Ukraine
- Misaki Kojima, Nagoya University, Japan
- René Thiemann, University of Innsbruck, Austria (co-chair)
- Vincent van Oostrom, University of Sussex, UK

CoCo Steering Committee

- Aart Middeldorp, University of Innsbruck, Austria
- Naoki Nishida, Nagoya University, Japan
- Teppei Saito, JAIST, Japan
- René Thiemann, University of Innsbruck, Austria
- Sarah Winkler, Free University of Bozen-Bolzano, Italy

Contents

Loop Elimination in Process Graphs is Confluent when Pruning Steps are Added

Clemens Grabmayer

Gran Sasso Science Institute,
L'Aquila, Abruzzo, Italy
clemens.grabmayer@gssi.it

Abstract

Process graphs that are interpretations of '1-free' regular expressions in Milner's process semantics for regular expressions have the Loop Existence and Elimination Property (LEE). Hereby a process graph satisfies LEE if the procedure of loop elimination, in which in every step a loop subgraph is decoupled by removing its entry transitions and then garbage collection is performed, terminates in a process graph without an infinite trace. Loop elimination in finite process graphs is terminating, but typically not confluent.

We explain that loop elimination can be turned into a confluent rewrite system on all process graphs by adding pruning steps that remove transitions to deadlocking states. For this purpose we perform a critical-pair like analysis that involves bi-loop subgraphs, and use the decreasing diagram method. This confluence result has two expedient consequences: for finite process graphs, LEE can be decided in polynomial time, and a layered version of LEE (no loops are eliminated from bodies of already eliminated ones) coincides with LEE.

We report on an aspect of work on the process semantics of regular expressions that concerns a procedure for analysing the structure of process graphs by decomposition. Formalising this procedure via rewrite relations helped us to clarify the situation. While the confluence result that we describe here does not require new techniques, it has some tangible consequences.

1 Introduction

Starting from a formalisation of regular (finite-state) processes as μ-terms, Milner (1984, [6]) defined an interpretation P of regular expressions e as regular processes $P(e)$. The interpretations of 0, letters a, sums $e_1 + e_2$, products $e_1 \cdot e_2$, and iterations e^* are as follows: $P(0)$ denotes deadlock, $P(a)$ performs a single action named "a" before terminating successfully, $P(e_1 + e_2)$ chooses to continue as $P(e_1)$ or $P(e_2)$, $P(e_1 \cdot e_2)$ executes $P(e_1)$, and upon its successful termination executes $P(e_2)$, and $P(e^*)$ iterates $P(e)$ (if it terminates successfully) arbitrarily often, but with the option to terminate successfully before each iteration. For 0^* we also use the constant 1 whose interpretation $P(1) = P(0^*)$ is the process that immediately terminates successfully. Based on this interpretation, Milner then defined the process semantics $[\![e]\!]_P$ of a regular expression e as the bisimilarity equivalence class $[P(e)]_{\leftrightarrows}$ of its process interpretation $P(e)$.

Unlike for the language interpretation of regular expressions, in which every language that is accepted by a finite-state automaton is the interpretation of a regular expression, not every finite-state process is bisimilar to the process interpretation of a regular expression. There are finite process graphs that are neither P-expressible (the interpretation of a regular expression) nor $[\![\cdot]\!]_P$-expressible (bisimilar to a P-expressible graph). In order to recognise, and reason about $[\![\cdot]\!]_P$-expressible graphs it was therefore desirable to characterise the structure of at least the P-expressible graphs. For this purpose Fokkink and I formulated the Loop Existence and Elimination Condition LEE [5], which checks the structure of a process graph by loop-elimination decomposition steps. These steps remove infinite-trace 'loop subgraphs', which correspond

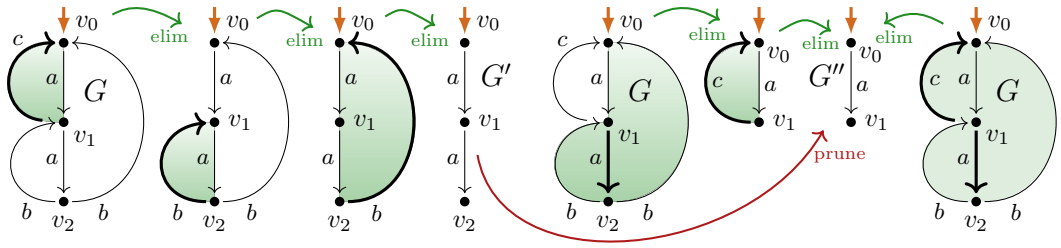


Figure 1: Three different $\to_{\mathrm{elim}}$ loop elimination sequences from the process interpretation $G = P(e)$ of the regular expression $e = ((1\cdot a)\cdot(c\cdot a + a\cdot(b+b\cdot a))^*)\cdot 0$. For every $\to_{\mathrm{elim}}$ step the eliminated loop subchart is indicated by drawing its loop-entry transition(s) in boldface, and colouring the area within its body. These rewrite sequences end in the normal form graphs G' and G'' neither of which permits an infinite trace. Therefore $G = P(e)$ satisfies LEE. But as G has two normal forms, $\to_{\mathrm{elim}}$ is not confluent. However, a $\to_{\mathrm{prune}}$ step can join the $\to_{\mathrm{elim}}$ fork. We note that none of the vertices of the graphs above permit immediate successful termination.

to interpretations $P(f^*)$ of iterations f^* for expressions f that are star-free and normed (i.e., $P(f)$ has a trace to successful termination). Loop subgraphs are peeled off from a given graph G one by one, as long as such steps are possible. If a normal form graph without an infinite trace is found, then LEE holds for G, and otherwise it fails. See Fig. 1 for a graph G that satisfies LEE. It can be shown (see [4]) that LEE characterises process interpretations of 'under-star-1-free' regular expressions (with restricted use of iterations within iterations), and that a generalisation 1-LEE [2] for graphs with 1-transitions (for empty steps) characterises process interpretations. In Sect. 2 we recapitulate the loop-elimination rewrite relation $\to_{\mathrm{elim}}$, and the property LEE.

Loop elimination steps can typically be performed in several ways, thereby 'parsing' a given P-expressible graph G differently while intuitively synthesising from G different regular expressions e with $P(e) = G$ or $P(e) \leftrightarrows G$. Typically $\to_{\mathrm{elim}}$ steps are not confluent, see Fig. 1 for an example. Here we report that a confluent rewrite system can be obtained by adding $\to_{\mathrm{prune}}$ steps that remove transitions to deadlock states. Such transitions cannot be part of infinite traces enabled by loop subgraphs, and therefore they do not hinder the applicability of $\to_{\mathrm{elim}}$ steps.

In Sect. 3 we give an outline for why $\to_{\mathrm{elim}}$ and $\to_{\mathrm{prune}}$ steps form a decreasing abstract rewriting system. For this purpose we focus attention on a lemma concerning $\to_{\mathrm{elim}}$ forks. The decreasing diagram method by van Oostrom [8, 7, 9] and de Bruijn then guarantees that $\to_{\mathrm{elim}} \cup \to_{\mathrm{prune}}$ is confluent. Finally in Sect. 4 we describe two significant consequences of this result. First, that LEE can be decided in polynomial time, and second, that a restriction of $\to_{\mathrm{elim}}$ for which only such loop subgraphs are eliminated whose start vertices are not in the body of previously eliminated loops defines a specialisation LLEE of LEE that is equivalent with LEE.

The exposition here is supplemented in a slightly extended version with appendix including some more of the proofs, available at `https://clegra.github.io/lf/le-conf-IWC-26.pdf`.

2 Loop Elimination and Pruning

A *(process) graph* (*with an immediate termination predicate*) is a tuple $G = \langle V, A, v_{\mathrm{s}}, \to, \Downarrow\rangle$ that consists of a set V of *vertices*, a set A of *actions*, a start vertex $v_{\mathrm{s}} \in V$ (hence $V \neq \varnothing$), a *(labelled) transition relation* $\to \subseteq V \times A \times V$, and a predicate $\Downarrow \subseteq V$ on the states that indicates immediate termination. We write $v_1 \xrightarrow{a} v_2$ for a transition $\langle v_1, a, v_2\rangle \in \to$, and $v\Downarrow$ for a state $v \in \Downarrow$ with immediate termination permitted, as well as $v\downarrow$ if $v \notin \Downarrow$. We say that a vertex $v \in V$

is *start-vertex connected* if there is a path from v_s to v. We usually assume, with the exception of the definition of rewrite steps, graphs to be such that all vertices are start-vertex connected.

Let $G = \langle V, A, v_s, \to, \Downarrow\rangle$ be a graph. We define the result of *garbage collection in* G based on the subset $V_0 \subseteq V$ of vertices of G that are start-vertex connected as $\mathsf{gc}(G) := \langle V_0, A, v_s, \to \cap (V_0 \times A \times V_0), \Downarrow \cap V_0\rangle$. Now let $v \in V$. By $G{\downarrow_*^v} := \mathsf{gc}(\langle V, A, v, \to, \Downarrow\rangle)$ we define the *subgraph of* G *at* v that results from G by changing the start vertex to v, and then performing garbage collection. Let again $v \in V$, and additionally $T \subseteq \langle v\to\rangle := \{\langle v, a, w\rangle \mid \langle v, a, w\rangle \in \to\}$ be a set of transitions from v in G. We define by $G{\downarrow_*^{v,T}} := \langle V_0, A, v, T \cup ((V_0 \setminus \{v\}) \times A \times V_0), \Downarrow \cap V_0\rangle$ the *generated subgraph of* G *from/until* v *with respect to* T where $V_0 \subseteq V$ is the set of vertices on traces in G that start in v, take a transition from T, and then continue onwards as long as possible but stop whenever v is reached again.

Definition 2.1 (loop graphs, loop subgraphs, bi-loops)**.** Let $G = \langle V, A, v_s, \to, \Downarrow\rangle$ be a graph.

We say that G is a *loop (process graph)* if it satisfies the following three conditions:

(L1) There is an infinite trace from the start vertex v_s of G (denoted by $\infty(v_s)$ and by $\infty(G)$).

(L2) Every infinite trace from the start vertex v_s of G returns to v_s.

(L3) Immediate termination is only permitted at the start vertex of G, i.e. $v\Downarrow$ only if $v = v_s$.

Then we call transitions from v_s *loop-entry transitions*, and the other ones *loop-body transitions*.

By a *loop subgraph* of G (where G does not have to be a loop itself) we mean a generated subgraph $G{\downarrow_*^{v,T}}$ of G from/until $v \in V$ with respect to $T \subseteq \langle v\to\rangle$ such that $G{\downarrow_*^{v,T}}$ is a loop.

Let $v_1, v_2 \in V$. We say that G is a *bi-loop with respect to* v_1, v_2 if $v_s = v_1 \neq v_2$, and G is a start-vertex connected loop chart that remains a start-vertex connected loop chart if its start vertex is changed to v_2, that is, for each $i \in \{1, 2\}$, $G{\downarrow_*^{v_i}} = \langle V, A, v_i, \to, \Downarrow\rangle$ (note the absence of garbage collection) is a start-vertex connected loop chart.

Note that for a bi-loop G with respect to v_1, v_2 it holds that $G = G{\downarrow_*^{v_1}}$, v_2 is reachable from v_1, and v_1 is reachable from v_2, in each of $G = G{\downarrow_*^{v_1}}$ and $G{\downarrow_*^{v_2}}$. The following lemma states that bi-loops can also be defined by three conditions similar to the loop chart conditions (L1)–(L3).

Lemma 2.2. *Let $B = \langle V, A, v_s, \to, \Downarrow\rangle$ be a graph, and let $v_1, v_2 \in V$ be such that $v_1 \neq v_2$.*

Then B is a bi-loop with respect to v_1, v_2 if and only if the following specialised variant of the loop conditions (L1)–(L3) hold: (B1) *there is an infinite trace from v_1,* (B2) *every infinite trace from v_1 visits v_2 before it visits v_1 again, and every infinite trace from v_2 visits v_1 before it visits v_2 again,* (B3) *no vertex of B permits immediate termination, i.e. $v\not\Downarrow$ for all $v \in V$.*

For defining decoupling $\mathsf{dcpl}(G, T)$ of sets T of transitions from G, and garbage collection $\mathsf{gc}(G)$ in G, we let $G = \langle V, A, v_s, \to, \Downarrow\rangle$ be a graph.

For a subset $T \subseteq \to$ of transitions of G, we define the (result of) *decoupling of* T *from* G as the graph $\mathsf{dcpl}(G, T) := \langle V, A, v_s, \to \setminus T, \Downarrow\rangle$ that results from G by removing the transitions in T.

Definition 2.3 (loop elimination $\to_{\mathrm{elim}}$, pruning $\to_{\mathrm{prune}}$, union $\to_{\mathrm{ep}}$)**.** We define loop elimination steps $\to_{\mathrm{elim}}$, and pruning steps $\to_{\mathrm{prune}}$ between graphs G and G' as follows:

$$G \to_{\mathrm{elim}} G' \;:\Longleftrightarrow\; \exists v \in V\, \exists T \subseteq \langle v\to\rangle \big[\, G{\downarrow_*^{v,T}} \text{ is a loop} \;\wedge\; G' = \mathsf{gc}(\mathsf{dcpl}(G, T))\,\big],$$
$$(\text{where } G = \langle V, A, v_s, \to, \Downarrow\rangle)$$

$$G \to_{\mathrm{prune}} G' \;:\Longleftrightarrow\; \exists t = \langle v, a, v'\rangle \in \to \big[\, \langle v'\to\rangle = \varnothing \;\wedge\; v'\not\Downarrow \;\wedge\; G' = \mathsf{gc}(\mathsf{dcpl}(G, \{t\}))\,\big].$$
$$(\text{where } G = \langle V, A, v_s, \to, \Downarrow\rangle)$$

By an $\to_{\mathrm{ep}}$ step we mean an $\to_{\mathrm{elim}}$ step or a $\to_{\mathrm{prune}}$ step. We will also use $\to_{\mathrm{elim}}$, $\to_{\mathrm{prune}}$, and $\to_{\mathrm{ep}}$ as suggestive notation for an(y) abstract rewriting system $\langle \mathcal{G}, \to\rangle$ with $\mathcal{G}$ a set of graphs that is closed under steps $\to$, and with rewrite relation $\to \in \{\to_{\mathrm{elim}}, \to_{\mathrm{prune}}, \to_{\mathrm{ep}}\}$.

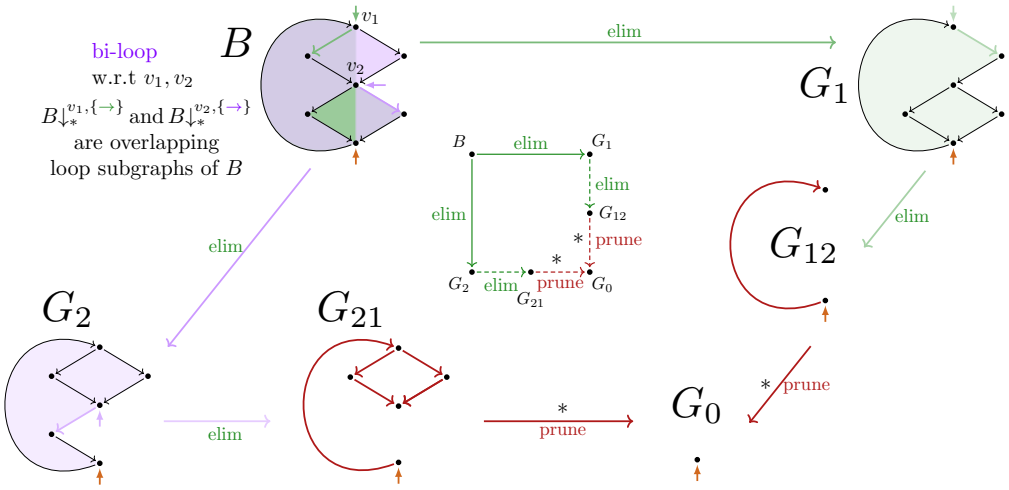


Figure 2: Prototypical bi-loop example justifying the elementary diagram (in the middle) for joining an elimination step fork $G_2 \leftarrow_{\text{elim}} B \rightarrow_{\text{elim}} G_1$ in the critical-pair like case of eliminating loop subgraphs with overlapping cyclic parts. The fork is joined by $\rightarrow_{\text{elim}}$ steps, and $\rightarrow_{\text{prune}}$ steps that 'eat up' the remaining bi-loop body: $G_2 \rightarrow_{\text{elim}} G_{21} \rightarrow^*_{\text{prune}} G_0 \leftarrow^*_{\text{prune}} G_{12} \leftarrow_{\text{elim}} G_1$.

Example 2.4. In Fig. 1 we illustrate, for a graph G, three maximal $\rightarrow_{\text{elim}}$ rewrite sequences, and a $\rightarrow_{\text{prune}}$ step that links the obtained $\rightarrow_{\text{elim}}$ normal forms. For each $\rightarrow_{\text{elim}}$ step the set T of loop-entry transitions that define the decoupled loop subchart is indicated in boldface. Two $\rightarrow_{\text{elim}}$ steps from a bi-loop with respect to overlapping loop subgraphs are depicted in Fig. 2.

Proposition 2.5. $\rightarrow_{\text{elim}}$ *is not confluent.* (See Fig. 1 for a counterexample to confluence.)

Definition 2.6 (LEE)**.** We say that a process graph G satisfies the *loop existence and elimination property* LEE, denoted by LEE(G), if G has an $\rightarrow_{\text{elim}}$ normal form graph that does not enable an infinite trace. More formally, we stipulate:

$$\text{LEE}(G) \;:\Longleftrightarrow\; \exists G_0\big(\,(\,G \rightarrow^*_{\text{elim}} G_0 \not\rightarrow_{\text{elim}}\,) \;\wedge\; \neg\infty(G_0)\big)\,.$$

Lemma 2.7. *On finite process graphs,* $\rightarrow_{\text{ep}} = \rightarrow_{\text{elim}} \cup \rightarrow_{\text{prune}}$ *is terminating.*

3 Loop Elimination with Pruning is Confluent

While the motivation for loop elimination stems from finite process graphs, we prove confluence of $\rightarrow_{\text{ep}}$ for all process graphs, including infinite ones. In order to obtain confluence of $\rightarrow_{\text{ep}}$ on only finite graphs it would suffice to show local confluence of $\rightarrow_{\text{ep}}$ (follows from Lem. 3.2 below), because Newman's lemma can be applied due to termination of $\rightarrow_{\text{ep}}$ in that case (Lem. 2.7). But we obtain confluence of $\rightarrow_{\text{ep}}$ on all graphs by showing that $\rightarrow_{\text{ep}} = \rightarrow_{\text{elim}} \cup \rightarrow_{\text{prune}}$ is decreasing, and by appealing to the Decreasing Diagram Theorem [8, 7]. Only the case of finding elementary diagrams for $\rightarrow_{\text{elim}}$ forks is non-trivial. We formulate that case as Lem. 3.1 below. It bears an apparent similarity with critical pair lemmas in term rewriting theory. For the crucial case of eliminating non-root overlapping loop subgraphs we illustrate a typical example in Fig. 2.

Lemma 3.1 (joining $\rightarrow_{\text{elim}}$ forks)**.** *Let* $G_2 \leftarrow_{\text{elim}} G \rightarrow_{\text{elim}} G_1$ *be a fork of* $\rightarrow_{\text{elim}}$ *steps in which the loop subgraphs* $G{\downarrow_*^{v_2,T_2}}$*, and* $G{\downarrow_*^{v_1,T_1}}$ *of* G *are eliminated, respectively.*

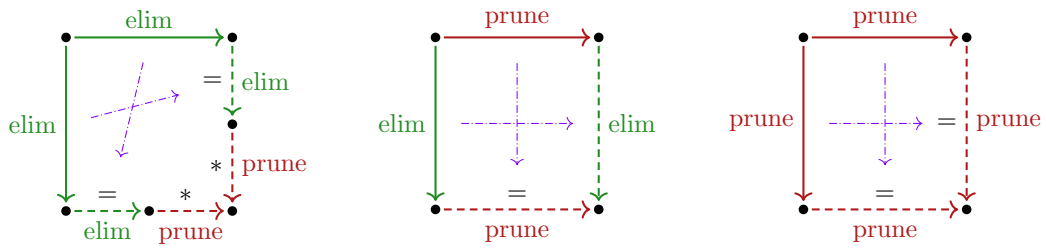


Figure 3: Elementary diagrams for showing that $\langle \mathcal{G}, \{\to_\ell\}_{\ell\in\{\mathrm{elim},\mathrm{prune}\}}\rangle$, for a set $\mathcal{G}$ of graphs that is closed under $\to_{\mathrm{elim}}$ and $\to_{\mathrm{prune}}$, is an abstract rewriting system that is decreasing with respect to precedence order prune $<$ elim. That these elementary diagrams are decreasing is indicated by the dash-dotted trace relation arrows.

Then this $\to_{\mathrm{elim}}$ fork can be joined appropriately with $\to_{\mathrm{prune}}$ steps towards a graph G_0 in the following three mutually exclusive, and exhaustive situations:

(i) Root-overlap*: $v_1 = v_2$, and both loops are contained in the loop $G{\downarrow_*^{v_1,T_1\cup T_2}}$. The $\to_{\mathrm{elim}}$ steps may overlap in their loop-entry transitions, but the fork can always be joined by eliminating residual loop-entries as follows: $G_2 \to_{\mathrm{elim}}^{=} \cdot \to_{\mathrm{prune}}^{*} G_0 \leftarrow_{\mathrm{prune}}^{*} \cdot \leftarrow_{\mathrm{elim}}^{=} G_1$.*

(ii) Parallel/nested*: $v_1 \neq v_2$, and the start-vertex connected cyclic parts $\circlearrowright(G{\downarrow_*^{v_1,T_1}})$ and $\circlearrowright(G{\downarrow_*^{v_2,T_2}})$ of the loops are vertex-disjoint. Then the $\to_{\mathrm{elim}}$ steps are independent of each other, except that the elimination of one loop may eliminate the other in the garbage collection operation. The fork can always be joined as follows: $G_2 \to_{\mathrm{elim}}^{=} G_0 \leftarrow_{\mathrm{elim}}^{=} G_1$.*

(iii) Non-root overlap *(see Fig. 2): $v_1 \neq v_2$, the start-vertex connected cyclic parts $\circlearrowright(G{\downarrow_*^{v_1,T_1}})$, and $\circlearrowright(G{\downarrow_*^{v_2,T_2}})$ have common vertices. The two loops together form a bi-loop w.r.t. v_1, v_2. By removing possibly remaining loop-entries at v_2 after the step $G \to_{\mathrm{elim}} G_2$, and at v_1 after the step $G \to_{\mathrm{elim}} G_1$, and then removing all transitions of the bi-loop with $\to_{\mathrm{prune}}$ steps, the fork can always be joined as follows: $G_2 \to_{\mathrm{elim}}^{=} \cdot \to_{\mathrm{prune}}^{*} G_0 \leftarrow_{\mathrm{prune}}^{*} \cdot \leftarrow_{\mathrm{elim}}^{=} G_1$.*

Lemma 3.2. *$\to_{\mathrm{ep}}$ is (locally) decreasing with respect to the family $\{\to_\ell\}_{\ell\in\{\mathrm{elim},\mathrm{prune}\}}$ of rewrite relations, with the precedence order prune $<$ elim.*

Proof. In Fig. 3 we illustrate the full set of elementary diagrams for the abstract rewriting system with the family $\{\to_\ell\}_{\ell\in\{\mathrm{elim},\mathrm{prune}\}}$ of rewriting relations.

The leftmost elementary diagram follows from the analysis of the three possible, mutually exclusive cases for $\to_{\mathrm{elim}}$ forks in Lem. 3.1. It describes the joining steps in the cases (i) and (iii) in Lem. 3.1, encompassing the situation in easier case (ii). The second elementary diagram in Fig. 3 concerning joining $\leftarrow_{\mathrm{prune}} \cdot \to_{\mathrm{elim}}$ forks by subcommutation of the two involved steps is due to the fact that transitions to deadlock states which are removed in $\to_{\mathrm{prune}}$ steps cannot be part of the infinite traces in the cyclic parts of loop subgraphs. The third elementary diagram in Fig. 3 expresses commutation of $\to_{\mathrm{prune}}$ steps: if the two $\to_{\mathrm{prune}}$ steps coincide, then they have the same targets, which are joined by empty steps; if the steps are different, then they commute.

All of these diagrams are decreasing with respect to the precedence order prune $<$ elim of reduction labels, as witnessed by the trace relation indicated by arrows in Fig. 3. □

Theorem 3.3. *$\to_{\mathrm{ep}} = \to_{\mathrm{elim}} \cup \to_{\mathrm{prune}}$ is confluent.*

Proof. From the fact that $\to_{\mathrm{ep}}$ is decreasing by Lem. 3.2 it follows by the Decreasing Diagrams Theorem (see [8, Cor. 3.9], [7, Thm. 14.2.8], [9, Thm. 1]) that $\to_{\mathrm{ep}}$ is confluent. □

Corollary 3.4. *When restricted to finite graphs,* $\to_{\mathrm{ep}} = \to_{\mathrm{elim}} \cup \to_{\mathrm{prune}}$ *is complete (that is,* $\to_{\mathrm{ep}}$ *confluent and terminating).*

Corollary 3.5. *Every finite graph has a unique* $\to_{\mathrm{ep}}$ *normal form graph.*

4 Consequences

From confluence of $\to_{\mathrm{ep}}$ by Thm. 3.3 we obtain the following corollary by utilising basic rewriting properties of $\to_{\mathrm{elim}}$ and $\to_{\mathrm{prune}}$ that are easy to establish: $\to_{\mathrm{prune}}$ postpones over $\to_{\mathrm{elim}}$, and $\to_{\mathrm{prune}}$ steps preserve as well as reflect $\to_{\mathrm{elim}}$ normal forms, and the properties $\infty(\cdot)$ and $\neg\infty(\cdot)$.

Corollary 4.1. *For every finite process graph* G *the following statements are equivalent:*

(i) LEE(G)*, that is, there is an* $\to_{\mathrm{elim}}$ *normal form of* G *without an infinite trace.*

(ii) The (by Cor. 3.5) unique $\to_{\mathrm{ep}}$ *normal form of* G *does not enable an infinite trace.*

(iii) Every $\to_{\mathrm{elim}}$ *normal form of* G *does not enable an infinite trace.*

Now Cor. 4.1, (i) $\Leftrightarrow$ (iii), entails that we can verify LEE(G), for every finite graph G, simply by constructing an arbitrary $\to_{\mathrm{elim}}$ rewrite sequence to a normal form G_0, and then confirming whether $\neg\infty(G_0)$ holds; if the test fails, we can conclude $\neg$LEE(G). As $\to_{\mathrm{elim}}$ rewrite sequences to normal form and the test are implementable in polynomial time, we obtain the next theorem.

Theorem 4.2. *For finite graphs,* LEE *is decidable in* PTIME *with the graph size as input size.*

For extracting regular expressions e with $P(e) \leftrightarrows G$ from $\to_{\mathrm{elim}}$ rewrite sequences that witness that a graph G satisfies LEE(G) it is expedient to only use 'layered' loop elimination $\overset{\circ}{\to}_{\mathrm{elim}}$ steps. In such steps, loops are never eliminated from bodies of already eliminated ones. Defining $\overset{\circ}{\to}_{\mathrm{elim}}$ requires history awareness, which can be formalised by vertex-marked versions ${}^{\bullet}G$ of graphs G in which the vertices in the body of already eliminated loops are marked.

Definition 4.3 (LLEE)**.** A graph G has the *layered* loop existence and elimination property LLEE, denoted by LLEE(G), if G has an $\overset{\circ}{\to}_{\mathrm{elim}}$ normal form graph without an infinite trace:

$$\mathrm{LLEE}(G) \;:\Longleftrightarrow\; \exists\, {}^{\bullet}G_0 \big(\big({}^{\circ}G \overset{\circ}{\to}{}^{*}_{\mathrm{elim}} {}^{\bullet}G_0 \not\overset{\circ}{\to}_{\mathrm{elim}} \big) \;\wedge\; \neg\infty({}^{\bullet}G_0) \big)\,.$$

Theorem 4.4. LLEE *coincides with* LEE.

Proof (Idea). For every sequence ${}^{\circ}G \overset{\circ}{\to}{}^{*}_{\mathrm{elim}} {}^{\bullet}G_1$ from an initially unmarked graph ${}^{\circ}G$ it holds that every $\to_{\mathrm{elim}}$ step from ${}^{\bullet}G_1$ that does not correspond to an $\overset{\circ}{\to}_{\mathrm{elim}}$ step gives rise to an $\overset{\circ}{\to}_{\mathrm{elim}}$ step from ${}^{\bullet}G_1$: the two steps eliminate overlapping loops in the same bi-loop, see Lem. 3.1, (iii). □

5 Conclusion

Three further questions that came up during our work on the confluence of $\to_{\mathrm{ep}}$, and its consequences are the following: (1) While a rough estimate (cf. version with appendix) of the complexity of deciding LEE according to Thm. 4.2 yields LEE $\in \mathrm{O}(|G|^3)$, there is much room for improvement. How fast can an algorithm for implementing $\to_{\mathrm{elim}}$, and deciding LEE be? (2) We want to analyse systematically how, for a graph G with LEE, any two regular expressions e_1, e_2 with $P(e_1) = P(e_2) \simeq G$ that can be extracted from two maximal $\to_{\mathrm{elim}}$ rewrite sequences from G (see [5, 1, 3]) can be proved equivalent with respect to $[\![\cdot]\!]_P$. (3) Just as postponement of $\overset{\circ}{\to}_{\mathrm{elim}}$ over $\to_{\mathrm{elim}}$ is closely connected to confluence of $\to_{\mathrm{ep}}$ (Thm. 3.3), we wonder how the decreasing diagram method (perhaps in its 'converted' version by van Oostrom [9]) can best be adapted in general for showing postponement of a family of rewrite relations over another family.

References

[1] Clemens Grabmayer. A Coinductive Version of Milner's Proof System for Regular Expressions Modulo Bisimilarity. In Fabio Gadducci and Alexandra Silva, editors, *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*, volume 211 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:23, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CALCO.2021.16.

[2] Clemens Grabmayer. Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete (Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions). In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, pages 1–13, New York, NY, USA, 2022. Association for Computing Machinery. Extended version as report arXiv:2209.12188 on arxiv.org, September, 2022. doi:10.1145/3531130.3532430.

[3] Clemens Grabmayer. A Coinductive Reformulation of Milner's Proof System for Regular Expressions Modulo Bisimilarity. *Logical Methods in Computer Science*, Volume 19, Issue 2, Jun 2023. URL: https://lmcs.episciences.org/11519, doi:10.46298/lmcs-19(2:17)2023.

[4] Clemens Grabmayer. Towards a Characterisation of the Graph Structure of Process Interpretations of Regular Expressions. Extended Abstract for TERMGRAPH 2026 (15th Int. Workshop on Computing with Terms and Graphs), published on the workshop's webpage http://termgraph.org.uk/2026, 2026. Also available at https://clegra.github.io/lf/pi-struc-TG-26.pdf.

[5] Clemens Grabmayer and Wan Fokkink. A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, page 465–478, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394744.

[6] Robin Milner. A Complete Inference System for a Class of Regular Behaviours. *Journal of Computer and System Sciences*, 28(3):439–466, 1984. doi:10.1016/0022-0000(84)90023-0.

[7] Terese. *Term Rewriting Systems*. Number 55 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003. doi:10.1017/S095679680400526X.

[8] Vincent van Oostrom. Confluence by Decreasing Diagrams. *Theoretical Computer Science*, 126(2):259–280, 1994. doi:10.1016/0304-3975(92)00023-K.

[9] Vincent van Oostrom. Confluence by Decreasing Diagrams (Converted). In Andrei Voronkov, editor, *Rewriting Techniques and Applications*, pages 306–320, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-70590-1_21.

Verifying and Generalizing Simultaneous Critical Pairs

René Thiemann

University of Innsbruck, Austria
rene.thiemann@uibk.ac.at

Abstract

Okui proved that simultaneous critical pairs (SCPs) can be used as a sufficient criterion to ensure confluence of term rewrite systems. His definitions, lemmas and proofs where reformulated by Kirk and Middeldorp. They heavily utilize proof terms and finally arrive at a formalized proof of Okui's result in Isabelle/HOL. However, Kirk and Middeldorp's formalization lacks an executable algorithm to compute the set of proof term based SCPs in a verified way.

In this work, we provide such an algorithm, and we further modify the formalization in such a way, that SCPs can also be used to show commutation, generalizing the confluence result. Our results have fully been integrated in the certifier CeTA, that now can deal with both commutation- and confluence-proofs by SCPs.

1 Introduction

Confluence is an important and well-studied property of term rewrite systems (TRSs). Since confluence is an undecidable property, several sufficient criteria have been developed, often using (variants of) critical pairs. For instance, there is van Oostrom's criterion of development closedness [9] using critical pairs, there is Toyama's criterion [8] based on parallel critical pairs, and there is Okui's criterion [6] based on simultaneous critical pairs (SCPs).

In previous works the mentioned confluence criteria have been formally verified in Isabelle/HOL [5]: Kirk and Middeldorp handled development closedness and Okui's criterion [4, 3], and Hirokawa et al. verified Toyama's criterion [2]. Here, during the formalization some of the criteria have been generalized from confluence to commutation. In this work, we extend this line of research in two directions.

- We generalize Okui's criterion from confluence to commutation, based on the formalization of Kirk and Middeldorp.
- We develop a verified implementation of SCPs, so that Okui's criterion can be checked by our certifier CeTA [7].

Regarding the second contribution, note that Kirk and Middeldorp provide a definition of SCPs that uses unbounded existential quantifiers, which makes an implementation challenging.

2 Preliminaries

We assume familiarity with term rewriting [1]. Given a binary relation $\to$, we write $\to^*$ for its reflexive closure, $\to^=$ for its reflexive closure, and $\leftarrow$ for its inverse. Given two binary relations $\to_1$ and $\to_2$, $\to_1 \circ \to_2$ denotes the relation composition of $\to_1$ and $\to_2$. Relations $\to_1$ and $\to_2$ commute if ${}^*_1\!\leftarrow \circ \to_2^* \subseteq \to_2^* \circ {}^*_1\!\leftarrow$, and they strongly commute if ${}_1\!\leftarrow \circ \to_2 \subseteq \to_2^= \circ {}^*_1\!\leftarrow$. Strong commutation implies commutation. Confluence of $\to$ is commutation where $\to_1 = \to_2 = \to$, and similarly strong confluence is strong commutation of two identical relations.

First order terms $s, t, \ell, r, \ldots \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ are build over some infinite set of variables $x, y, z, \ldots \in \mathcal{V}$ and some set of function symbols $a, b, f, g, \ldots \in \mathcal{F}$, where each $f \in \mathcal{F}$ has a fixed arity. We assume that $\mathcal{F}$ and $\mathcal{V}$ are arbitrary but fixed within this paper. A substitution σ is a function of type $\mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$, and we write $t\sigma$ for the term that is obtained by replacing each variable x in term t by $\sigma(x)$. A context C is a term with exactly one hole, and $C[t]$ is the term where this hole is replaced by t. Let $\mathcal{V}ars(t)$ denote the set of variables of term t. Given a term t and a position p within this term, $t|_p$ denotes the subterm of t at position p and $t[s]_p$ replaces the subterm at position p by s. For every term t, we use the notation $FPos(t)$ for the set of positions p of t that satisfy $t|_p \notin \mathcal{V}$.

A term rewrite system (TRS) $\mathcal{R}$ is a set of rules $\ell \to r$ where $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, $\mathcal{V}ars(\ell) \supseteq \mathcal{V}ars(r)$, and $\ell \notin \mathcal{V}$. A TRS $\mathcal{R}$ is left-linear if for every rule $\ell \to r \in \mathcal{R}$ the term ℓ is linear, i.e., no variable occurs more than once in ℓ. Given a TRS $\mathcal{R}$, its corresponding rewrite relation is defined as $s \to_{\mathcal{R}} t$ iff $s = C[\ell\sigma]$ and $t = C[r\sigma]$ for some rule $\ell \to r \in \mathcal{R}$, some context C and substitution σ. A TRS $\mathcal{R}$ is confluent if $\to_{\mathcal{R}}$ is confluent, and TRSs $\mathcal{R}$ and $\mathcal{S}$ commute if $\to_{\mathcal{R}}$ and $\to_{\mathcal{S}}$ commute. The multistep relation $\multimap_{\mathcal{R}}$ of a TRS is defined as the least relation such that $x \multimap_{\mathcal{R}} x$, $f(s_1, \ldots, s_n) \multimap_{\mathcal{R}} f(t_1, \ldots, t_n)$ whenever $s_i \multimap_{\mathcal{R}} t_i$ for all i, and $\ell\sigma \multimap_{\mathcal{R}} r\delta$ whenever $\ell \to r \in \mathcal{R}$ and $\sigma(x) \multimap_{\mathcal{R}} \delta(x)$ for all $x \in \mathcal{V}ars(\ell)$. It is known that $\to_{\mathcal{R}} \subseteq \multimap_{\mathcal{R}} \subseteq \to_{\mathcal{R}}^*$ for every TRS $\mathcal{R}$. Therefore, strong confluence of $\multimap_{\mathcal{R}}$ implies confluence of $\mathcal{R}$. Similarly, strong commutation of $\to_{\mathcal{S}}$ and $\multimap_{\mathcal{R}}$, which is equivalent to strong commutation of $\multimap_{\mathcal{S}}$ and $\multimap_{\mathcal{R}}$, implies commutation of $\mathcal{R}$ and $\mathcal{S}$.

Okui's criterion demands that a TRS $\mathcal{R}$ is left-linear and all SCPs $s \mathrel{\multimap\!\!\leftarrow}_{\mathcal{R}} \circ \to_{\mathcal{R}} t$ are joinable in a certain way, i.e., $s \to_{\mathcal{R}}^* \circ \mathrel{\leftarrow\!\!\multimap}_{\mathcal{R}} t$. He proved that for such TRSs, $\multimap_{\mathcal{R}}$ is strongly confluent and thus, $\mathcal{R}$ is confluent. Kirk and Middeldorp formalized this result and gave a definition of SCPs that is based on proof terms, cf. Definition 2.

A proof term of a TRS $\mathcal{R}$ is a term $A, B, \ldots \in \mathcal{T}(\mathcal{F} \uplus \mathcal{R}, \mathcal{V})$, i.e., where function symbols can either be normal function symbols in $\mathcal{F}$ or rule symbols $\alpha, \beta, \ldots$ where each rule symbol represents a rewrite rule $\ell \to r \in \mathcal{R}$. Here, the arity of a rule symbol α is exactly the number of variables in the corresponding rule. There is a one-to-one correspondence between multisteps and proof terms, where *src* and *tgt* are operations that project rule symbols to their left-hand sides and right-hand sides, respectively. We say that a proof term is empty, if it does not contain any rule symbols, i.e., the corresponding multistep does not perform a single rewrite step. Merging two compatible proof terms is performed with a $\sqcup$ operation, that represents the union of the the individual rewrite steps. The operation $RP(A)$ provides a distinct list of redex patterns of a proof term A, where pattern (α, p) is part of the list, if rule α is applied at position p in the step $src(A) \multimap tgt(A)$ that is encoded by A. The list is sorted by positions in ascending order. The function $OV(A, B)$ defines a new proof term that is similar to A, however, only those redex patterns of A are kept in $OV(A, B)$ that overlap with redexes of B.

Example 1. *If $\mathcal{R}$ consists of rules $\alpha : a \to b$ and $\beta : f(h(x), y) \to g(y, c, x)$, then $f(\underline{f(h(\underline{a}), a)}, z) \multimap_{\mathcal{R}} f(\underline{g(a, c, \underline{b})}, z)$ and this step is encoded in the proof term $A := f(\beta(\alpha, a), z)$, i.e., $src(A) = f(f(h(a), a), z)$ and $tgt(A) = f(g(a, c, b), z)$. Proof terms A and $B := f(f(h(\alpha), \alpha), z)$ are compatible and $A \sqcup B = f(\beta(\alpha, \alpha), z)$ with $tgt(A \sqcup B) = f(g(b, c, b), z)$. The lists of redex patterns in this example are $RP(A) = [(\beta, 1), (\alpha, 1.1.1)]$ and $RP(B) = [(\alpha, 1.1.1), (\alpha, 1.2)]$. Since $(\beta, 1) \in RP(A)$ does not overlap with redexes in B we obtain $RP[OV(A, B)] = [(\alpha, 1.1.1)]$ and $OV(A, B) = f(f(h(\alpha), a), z)$.*

The upcoming definition of SCPs is taken literally from Kirk and Middeldorp. Note that our main interest in this paper is to provide a verified implementation of SCPs. For the question why SCPs are defined in the way they are, we refer to the motivation given in the original papers [3, 6].

Definition 2 (Simultaneous Critical Pairs using Proof Terms [3, Definition 5.2]). *Let $\mathcal{R}$ be some TRS. We define $SCP(\mathcal{R})$ as the set of all pairs (A, B) of proof terms A and B of $\mathcal{R}$ that satisfy the following conditions for suitable values of $\alpha_1, p_1, \beta, p, \tau, \ldots$.*

1. $RP(A) = [(\alpha_1, p_1), \ldots, (\alpha_n, p_n)]$
2. $RP(B) = [(\beta, q)]$
3. $q = \varepsilon$ *or* $p_1 = \varepsilon$
4. $OV(A, B) = A$
5. *there are renaming substitutions $\sigma_{\alpha_1}, \ldots, \sigma_{\alpha_n}$ and σ_β such that all variables of rules $\alpha_1, \ldots, \alpha_n, \beta$ are renamed apart*
6. $\ell = lhs(\alpha_1)\sigma_{\alpha_1}[lhs(\beta)\sigma_\beta]_q$
7. τ *is a most general unifier of* $\{lhs(\alpha_1)\sigma_{\alpha_1} \approx \ell|_{p_1}, \ldots, lhs(\alpha_n)\sigma_{\alpha_n} \approx \ell|_{p_n}\}$
8. $A = \bigsqcup_{i=1}^{n} A_i$ *where* $A_i = \ell\tau[\alpha_i(\sigma_{\alpha_i}\tau)]_{p_i}$ *for all* $1 \leq i \leq n$
9. $B = \ell\tau[\beta(\sigma_\beta\tau)]_q$

Here, $lhs(\alpha)$ returns the left-hand side of a rule symbol α. The conditions in particular enforce $n \geq 1$, so proof term A must be non-empty.

Theorem 3 (Okui's Criterion using Proof Terms [3, Listing 3]). *If $\mathcal{R}$ is left-linear and $tgt(A) \to_{\mathcal{R}}^* \circ \mathrel{\text{-}\!\circ\!\!\leftarrow}_{\mathcal{R}} tgt(B)$ for all $(A, B) \in SCP(\mathcal{R})$, then $\mathrel{\text{-}\!\circ\!\!\to}_{\mathcal{R}}$ is strongly confluent and thus, $\mathcal{R}$ is confluent.*

3 From Confluence to Commutation

Our first aim is to try to generalize Definition 2 and Theorem 3 to commutation. Basically, an SCP (A, B) encodes the critical peak $tgt(A)\ {}_{\mathcal{R}}\!\mathrel{\text{-}\!\circ\!\!\leftarrow}\ src(A) = src(B) \to_{\mathcal{R}} tgt(B)$. In order to generalize this setting to commutation, we just have replace one of the TRSs in the peak by another TRS $\mathcal{S}$:

$$tgt(A)\ {}_{\mathcal{R}}\!\mathrel{\text{-}\!\circ\!\!\leftarrow}\ src(A) = src(B) \to_{\mathcal{S}} tgt(B).$$

To accommodate for this change we just have to generalize the definition of $SCP(\mathcal{R})$ to be based on two TRSs.

Definition 4 (SCPs for commutation of $\mathcal{R}$ and $\mathcal{S}$). *We define $SCP(\mathcal{R}, \mathcal{S})$ in the same way as $SCP(\mathcal{R})$ in Definition 2 with the only exception that instead of requiring that both A and B are proof terms of $\mathcal{R}$, we now demand that A is a proof term of $\mathcal{R}$ and B is a proof term of $\mathcal{S}$.*

Theorem 5 (Okui's Criterion for Commutation). *If $\mathcal{R}$ and $\mathcal{S}$ are left-linear and $tgt(A) \to_{\mathcal{S}}^* \circ\ {}_{\mathcal{R}}\!\mathrel{\text{-}\!\circ\!\!\leftarrow}\ tgt(B)$ for all $(A, B) \in SCP(\mathcal{R}, \mathcal{S})$, then $\to_{\mathcal{S}}$ and $\mathrel{\text{-}\!\circ\!\!\to}_{\mathcal{R}}$ strongly commute and thus, $\mathcal{R}$ and $\mathcal{S}$ commute.*

In order to arrive at this new theorem, we heavily used the existing formalization: It just took two hours to replace each occurrence of$\mathcal{R}$ by either $\mathcal{R}$ or $\mathcal{S}$ in the existing formalization,

i.e., we needed to make these replacement in every intermediate definition, every lemma and in every proof. The property that we made the correct replacements, and that every generalized statement is still provable was automatically checked by Isabelle: we did not encounter a single problem during the generalization process.

First note that Theorem 5 can easily be extended to show that joinability of SCPs actually characterizes strong commutation of $\multimap_{\mathcal{S}}$ and $\multimap_{\mathcal{R}}$. The reason is that one direction is immediate: since every SCP (A, B) encodes a peak $tgt(A)\ {}_{\mathcal{R}}\!\!\leftarrow\!\!\circ\ \circ\ \multimap_{\mathcal{S}} tgt(B)$, all these peaks are joinable in the desired form, provided that $\multimap_{\mathcal{S}}$ and $\multimap_{\mathcal{R}}$ strongly commute.

Corollary 6 (Strong Confluence and Strong Commutation via Simultaneous Critical Pairs)**.** *Let $\mathcal{R}$ and $\mathcal{S}$ be left-linear TRSs.*

- *The relations $\multimap_{\mathcal{S}}$ and $\multimap_{\mathcal{R}}$ strongly commute if and only if $tgt(A) \to_{\mathcal{S}}^* \circ\ {}_{\mathcal{R}}\!\!\leftarrow\!\!\circ\ tgt(B)$ for all $(A, B) \in SCP(\mathcal{R}, \mathcal{S})$.*
- *The relation $\multimap_{\mathcal{R}}$ is strongly confluent if and only if $tgt(A) \to_{\mathcal{R}}^* \circ\ {}_{\mathcal{R}}\!\!\leftarrow\!\!\circ\ tgt(B)$ for all $(A, B) \in SCP(\mathcal{R}, \mathcal{R})$.*

As a consequence of the corollary, we see that joinability of simultaneous critical pairs subsumes every criterion for left-linear TRSs that proves strong confluence (or strong commutation) of the multistep relation(s), e.g., the criterion of development closedness.

Further note that Theorem 5 is not symmetric, i.e., when actually trying to prove commutation of $\mathcal{R}$ and $\mathcal{S}$, one should try to apply Theorem 5 for $\mathcal{R}$ and $\mathcal{S}$ as stated, but also in the setting where the roles of $\mathcal{R}$ and $\mathcal{S}$ are swapped.

Finally note that for Okui's confluence criterion it is not known whether one can restrict Definition 2 in such a way that the single step is always at the root position (as it is done for parallel critical pairs), i.e., replace the disjunction $q = \varepsilon \vee p_1 = \varepsilon$ in Condition 3 just by $q = \varepsilon$. At least for commutation, we show that such a modification is not possible.

Example 7. *Let $\mathcal{R} = \{\alpha : f(a) \to b\}$ and $\mathcal{S} = \{\beta : a \to c\}$. These TRSs do not commute since $b\ {}_{\mathcal{R}}\!\leftarrow f(a) \to_{\mathcal{S}} f(c)$, and b and $f(c)$ are not joinable.*

If however we replace Condition 3 by $q = \varepsilon$, then $SCP(\mathcal{R}, \mathcal{S})$ must be empty. Otherwise, there would be some $(A, B) \in SCP(\mathcal{R}, \mathcal{S})$. But this implies that B must be β, i.e., $src(A) = src(B) = a$, and hence A must be an empty proof term, violating the conditions in Definition 4. But if $SCP(\mathcal{R}, \mathcal{S}) = \emptyset$ with the modification, it shows that Theorem 5 becomes unsound.

4 Implementing Simultaneous Critical Pairs

Our main aim was to be able to use the existing formalization for enabling support of Okui's criterion in CeTA. To this end, two tasks have to be solved: first, one needs a verified algorithm that actually computes $SCP(\mathcal{R})$ (or $SCP(\mathcal{R}, \mathcal{S})$ for commutation), and second one needs to check the joining sequences that are provided by automated confluence and commutation tools. Checking left-linearity of $\mathcal{R}$ and $\mathcal{S}$ is trivial.

For the second task, all the machinery is already included in CeTA, i.e., tools just need to output for each simultaneous critical pair (s, t) a joining sequence of terms $s \multimap u_1 \multimap u_2 \multimap \ldots \multimap u_n \leftarrow\!\!\circ\ t$, so that $s \to^* u_n \leftarrow\!\!\circ\ t$ can be concluded. Here, CeTA takes care of mismatches in names of variables in SCPs, e.g., by using different renaming substitutions, etc. Moreover, trivial critical pairs (t, t) can be omitted in certificates, too.

Hence, it remains to solve the first task, an implementation of Definition 2 (or the more general Definition 4) for finite TRSs. For presentation reasons we only speak about Definition 2 in the sequel, though the formalization covers Definition 4.

The first step we performed, is to actually change Definition 2, so that $SCP(\mathcal{R})$ becomes finite for finite TRSs. Definition 2 has the problem that there are infinitely many possible SCPs which just differ by a variable renaming. The problem is that with the existential quantifier one has to consider all renaming substitutions, and also every most general unifier τ. This problem is already partially solved in the Isabelle theories: there, the renaming substitutions are not arbitrary, but they are uniquely computed by some renaming algorithm, and this renaming algorithm is a parameter of the SCP definition. To deal with unifiers, we changed Condition 7 in a way that also the most general unifier τ is not an arbitrary one, but it is the single unifier that one gets from a fixed unification algorithm. Fortunately, this change did not require any big adjustments in the formal proof of Theorem 5.

Note that it is not immediately clear how to resolve the remaining existential quantifiers in Definition 2. For instance, A is defined via A_i where A_i can be computed only if (α_i, p_i) is known (Condition 8), and (α_i, p_i) is obtained from A (Condition 1). In order to break this cyclic dependence, we show that for finite TRSs there are only finitely many possibilities to choose the values of α_i and p_i. To this end, we use an alternative characterization of Condition 4 based on positions: condition $OV(A, B) = A$ is equivalent to

$$\forall 1 \le i \le n.\ \{p_i p \mid p \in FPos(lhs(\alpha_i))\} \cap \{qp \mid p \in FPos(lhs(\beta))\} \neq \emptyset.$$

Using this equivalence it will be possible to enumerate all possible redex patterns in Conditions 1 and 2 without knowing A and B.

Let us consider the easier case $q = \varepsilon$ first. Here, Condition 2 can be simplified to $q = \varepsilon \wedge \beta \in \mathcal{R}$, so an enumeration over all finitely many rules of $\mathcal{R}$ is possible to cover all possibilities to choose q and β. Since $q = \varepsilon$, condition $OV(A, B) = A$ simplifies further to

$$\forall 1 \le i \le n.\ p_i \in FPos(lhs(\beta)).$$

Hence, also for $RP(A)$ there are only finitely many possibilities, since the positions are taken from a finite set and there are only finitely many rule symbols. Instead of blindly enumerating all these possibilities, we use the following algorithm rp that takes a term t as input and returns a set of lists of redex patterns, namely those lists that correspond to proof terms where all redex patterns overlap with function positions of t. Here, $\oplus$ denotes list concatenation and $\mathit{unify\text{-}vd}(s, t)$ denotes that s and t unify after making the variables of these terms disjoint.

- $rp(x) = \{[\,]\}$
- $rp(f(t_1, \ldots, t_n)) = \Big(\bigcup_{ps_1 \in rp(t_1), \ldots, ps_n \in rp(t_n)} \bigoplus_{1 \le i \le n} [(\beta, i.p) \mid (\beta, p) \in ps_i]\Big)$
  $\cup \bigcup_{\alpha:\ell \to r \in \mathcal{R}, \mathit{unify\text{-}vd}(\ell, f(t_1, \ldots, t_n))} \mathit{rp\text{-}root}(f(t_1, \ldots, t_n), \alpha, \ell)$
- $\mathit{rp\text{-}root}(t, \alpha, \ell) = \bigcup_{ps_1 \in \mathit{rp\text{-}sub}(t,p_1), \ldots, ps_n \in \mathit{rp\text{-}sub}(t,p_n)} [(\alpha, \varepsilon)] \oplus \bigoplus_{1 \le i \le n} [(\alpha, p_i.p) \mid (\alpha, p) \in ps_i]$
  where $p_1, \ldots, p_n$ are the variable positions of ℓ
- $\mathit{rp\text{-}sub}(t, p) = rp(t|_p)$, if $p \in FPos(t)$, and $\mathit{rp\text{-}sub}(t, p) = \{[\,]\}$, otherwise

In the equation for $rp(f(t_1, \ldots, t_n))$ on the left of the $\cup$-operation we compute the redex patterns for those multisteps that do not perform a root step, and on the right of the $\cup$-operation are those redex patterns that include a root step. Here, the unification condition restricts the set of potential rules. For the root step itself, in $\mathit{rp\text{-}root}$ we insert the redex pattern (α, ε) for the root step at the front of the list, and then gather further redex patterns that are below variable positions of the left-hand side ℓ. Note that the case analysis in the definition of $\mathit{rp\text{-}sub}(t, p)$ is essential, since p is a variable position of ℓ, and this is not necessarily a position of t.

The algorithm rp and its auxiliary algorithms terminate, and it is characterized as follows.

Theorem 8 (Soundness of rp)**.** *Let $\mathcal{R}$ be left-linear. For every list ps and term t the following two statements are equivalent.*

- $ps \in rp(t)$.
- *There is some proof term A of $\mathcal{R}$ such that*

$$RP(A) = ps \wedge (\forall(\alpha, p) \in ps.\ p \in FPos(t) \wedge \mathit{unify\text{-}vd}(t|_p, lhs(\alpha))).$$

With the help of Theorem 8 we proved that SCPs with $q = \varepsilon$ can be computed as follows:

- Pick every rule symbol β of $\mathcal{R}$ and fix $q = \varepsilon$.
- Pick every non-empty list $[(\alpha_1, p_1), \ldots, (\alpha_n, p_n)]$ of $rp(lhs(\beta))$.
- Compute the remaining objects of Definition 2 following Condition 5 onwards.

For the case $q \neq \varepsilon$ we perform the following steps to compute the SCPs, again based on rp.

- Pick every rule $\alpha_1 : \ell_1 \to r_1 \in \mathcal{R}$ and fix $p_1 = \varepsilon$.
- Pick every $q \in FPos(\ell_1)$ satisfying $q \neq \varepsilon$.
- Pick every rule $\beta \in \mathcal{R}$ such that $\mathit{unify\text{-}vd}(lhs(\beta), \ell_1|_q)$.
- For each q_i from variable positions $q_2, \ldots, q_k$ of α_1, pick every qs_i from $rp(lhs(\beta)|_p)$ and define $ps_i = [(\gamma, q_i p') \mid (\gamma, p') \in qs_i]$ if $q_i = qp$ and $p \in FPos(lhs(\beta))$ for some p, and define $ps_i = [\,]$, otherwise.
- Define $(\alpha_2, p_2), \ldots, (\alpha_n, p_n)$ such that $[(\alpha_2, p_2), \ldots, (\alpha_n, p_n)] = \bigoplus_{i=2}^{k} ps_i$.
- Compute the remaining objects of Definition 2 following Condition 5 onwards.

The function *sim_cp_impl* of the IsaFoR library implements all these steps (using lists instead of sets) in the general case of Definition 4, and we verify that it exactly computes $SCP(\mathcal{R}, \mathcal{S})$. Moreover, the implementation is optimal in the sense that whenever the list representations of $\mathcal{R}$ and $\mathcal{S}$ are distinct, then so is $sim\_cp\_impl(\mathcal{R}, \mathcal{S})$. In particular, no critical pair is computed twice, since the implementation does not perform any explicit elimination of duplicates. We further tuned the implementation in a way that certain elements are precomputed, e.g., the variable positions of each left-hand side are only computed once.

We also integrated certificate generation for Okui's criterion in CSI [10], and because of our efforts, CSI is now able to produce certified proofs for three TRSs from the ARI-COPS database (# 419, # 463, and # 464), where prior to this work no certified proof was known. The generated proofs are verified by CeTA in a fraction of a second.

Both the formalization and the certified confluence proofs for the three TRSs are available on the following website. The website further includes links to the formalized definitions and theorems of this paper.

https://isafor-ceta.uibk.ac.at/experiments/iwc2026/

Acknowledgments

We thank the anonymous reviewers for their detailed and constructive feedback. This research was funded in part by the Austrian Science Fund (FWF) [Grant 10.55776/I5943].

References

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

[2] Nao Hirokawa, Dohan Kim, Kiraku Shintani, and René Thiemann. Certification of confluence- and commutation-proofs via parallel critical pairs. In Amin Timany, Dmitriy Traytel, Brigitte Pientka, and Sandrine Blazy, editors, *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*, pages 147–161. ACM, 2024. doi:10.1145/3636501.3636949.

[3] Christina Kirk and Aart Middeldorp. Formalizing simultaneous critical pairs for confluence of left-linear rewrite systems. In Kathrin Stark, Amin Timany, Sandrine Blazy, and Nicolas Tabareau, editors, *Proceedings of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2025, Denver, CO, USA, January 20-21, 2025*, pages 156–170. ACM, 2025. doi:10.1145/3703595.3705881.

[4] Christina Kohl and Aart Middeldorp. A formalization of the development closedness criterion for left-linear term rewrite systems. In Robbert Krebbers, Dmitriy Traytel, Brigitte Pientka, and Steve Zdancewic, editors, *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023*, pages 197–210. ACM, 2023. doi:10.1145/3573105.3575667.

[5] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-45949-9.

[6] Satoshi Okui. Simultaneous critical pairs and church-rosser property. In Tobias Nipkow, editor, *Rewriting Techniques and Applications, 9th International Conference, RTA-98, Tsukuba, Japan, March 30 - April 1, 1998, Proceedings*, Lecture Notes in Computer Science, pages 2–16. Springer, 1998. URL: https://doi.org/10.1007/BFb0052357, doi:10.1007/BFB0052357.

[7] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, Lecture Notes in Computer Science, pages 452–468. Springer, 2009. doi:10.1007/978-3-642-03359-9_31.

[8] Yoshihito Toyama. On the Church-Rosser property of term rewriting systems. *NTT ECL Technical Report (Japanese)*, 17672, 1981.

[9] Vincent van Oostrom. Development closed critical pairs. In Gilles Dowek, Jan Heering, Karl Meinke, and Bernhard Möller, editors, *Higher-Order Algebra, Logic, and Term Rewriting, Second International Workshop, HOA '95, Paderborn, Germany, September 21-22, 1995, Selected Papers*, Lecture Notes in Computer Science, pages 185–200. Springer, 1995. doi:10.1007/3-540-61254-8_26.

[10] Harald Zankl, Bertram Felgenhauer, and Aart Middeldorp. CSI - A confluence tool. In Nikolaj S. Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, Lecture Notes in Computer Science, pages 499–505. Springer, 2011. doi:10.1007/978-3-642-22438-6_38.

Confluence of Orthogonal Deterministic Higher-Order Pattern Rewrite Systems

Johannes Niederhauser and Aart Middeldorp

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

Abstract

We generalize the confluence result for orthogonal higher-order pattern rewrite systems to higher-order rewrite systems whose left hand sides consist of Yokoyama et al.'s deterministic higher-order patterns.

1 Introduction

Nipkow's higher-order rewriting systems (HRSs) [11] are popular in higher-order confluence research [4,7,13]. In particular, the restriction of the rules' left-hand sides to Miller's patterns [8] has proven to be very useful: The corresponding class of higher-order pattern rewrite systems (PRSs) exhibits a definition of critical pairs which behaves much like its first-order counterpart. For example, confluence of finite and terminating PRSs is decidable and orthogonal PRSs are confluent [7]. Recently, we introduced deterministic higher-order pattern rewrite systems (DPRSs) [9], a subclass of HRSs which generalizes PRSs by allowing Yokoyama et al.'s deterministic higher-order patterns (DHPs) [15] as left-hand sides of rules. In contrast to first-order term rewrite systems and PRSs, we also have to consider overlaps at variable positions for DPRSs. In this paper, we improve our initial definition of critical pairs of DPRSs given in [9]. This allows us to extend the confluence result for orthogonal PRSs to DPRSs.

2 Preliminaries

For a binary relation denoted by $\to$ we write $\leftarrow$ and $\to^*$ for its inverse and transitive-reflexive closure, respectively. Throughout this text, we will denote a sequence $a_1, \ldots, a_n$ by $\overline{a}_n$ where $n \geqslant 0$ and the corresponding set by $\{\overline{a}_n\}$. For every a, $\overline{a}_0$ represents the empty sequence which we denote by $()$. We implicitly enclose sequences in parentheses whenever it is necessary. For a binary relation R, $\overline{a}_n \; R \; \overline{b}_n$ abbreviates $a_1 \; R \; b_1, \ldots, a_n \; R \; b_n$. Similarly, for a (postfix) function f, $f(\overline{a}_n)$ $(\overline{a}_n f)$ denotes the pointwise application of f to $\overline{a}_n$. Given two sequences $\overline{a}_n$ and $\overline{b}_m$, their concatenation is written as $\overline{a}_n, \overline{b}_m$.

In this paper, we consider Nipkow's HRSs [7] which use simply-typed λ-terms in $\beta\eta$-long normal form [2,3]. Instead of viewing HRSs as rewrite systems modulo $\beta\eta$, we follow another tradition in higher-order rewriting where terms stay in their canonical form at all times due to the usage of hereditary substitution application [4,6]. Let $\mathcal{S}$ be a set of *sorts* (a, b).[1] The set $\mathcal{T}$ of *types* (σ, τ) is defined as follows: $\mathcal{S} \subseteq \mathcal{T}$ and if $\sigma_1, \ldots, \sigma_n \in \mathcal{T}$ and $a \in \mathcal{S}$ then $\overline{\sigma}_n \to a \in \mathcal{T}$ where we identify $() \to a$ with a. Suppose there is an infinite set $\mathcal{V}$ of typed variables (x, y, z, w) and a set $\mathcal{F}$ of typed function symbols (f, g) called *signature*, such that $\mathcal{V}$, $\mathcal{F}$, $\mathcal{S}$ and $\{\to\}$ are pairwise disjoint and there are infinitely many variables of each type. A *head* h is either

[1] This means that in the remainder, a and b are used exclusively to represent arbitrary sorts. We will continue to define naming conventions in this fashion.

a function symbol or a variable and we write $h : \sigma$ to denote that it has type σ. In order to keep the notation concise, we drop the symbol λ in abstractions. The following inference rule defines the set $\mathsf{Term}(\mathcal{F}, \mathcal{V}, \sigma)$ of *terms* (s, t, u, v) of type σ

$$\frac{h : \overline{\sigma}_n \to a \in \mathcal{F} \cup \mathcal{V} \quad \overline{t}_n \in \mathsf{Term}(\mathcal{F}, \mathcal{V}, \overline{\sigma}_n) \quad \overline{x}_k : \overline{\tau}_k \in \mathcal{V}^k}{\overline{x}_k.h(\overline{t}_n) \in \mathsf{Term}(\mathcal{F}, \mathcal{V}, \overline{\tau}_k \to a)}$$

where we use $\mathsf{Term}(\mathcal{F}, \mathcal{V}, \overline{\sigma}_n)$ as a shorthand for the set $\mathsf{Term}(\mathcal{F}, \mathcal{V}, \sigma_1) \times \cdots \times \mathsf{Term}(\mathcal{F}, \mathcal{V}, \sigma_n)$. We abbreviate $h()$ by h, $().s$ by s and follow the convention that postfix operations $\diamondsuit$ bind stronger than binders in terms, so $\overline{x}_n.t\diamondsuit = \overline{x}_n.(t\diamondsuit)$. The set $\mathsf{Term}(\mathcal{F}, \mathcal{V}, \mathcal{S})$ denotes the set of all terms which coincides with the set of well-typed λ-terms over $\mathcal{F}$ and $\mathcal{V}$ using sorts from $\mathcal{S}$ in $\beta\eta$-long normal form. As for function symbols and variables, we denote that a term s has type σ by $s : \sigma$. Given a term s, its set of free variables $\mathsf{FV}(s)$ is defined as usual. For a sequence of terms $\overline{s}_n$, we use $\mathsf{FV}(\overline{s}_n)$ as a shorthand for $\bigcup\{\mathsf{FV}(s_i) \mid 1 \leqslant i \leqslant n\}$. A term is *linear* if no free variable occurs more than once in it. Terms are uniquely defined up to the renaming of bound variables (α-renaming). Note that $\mathcal{V} \subseteq \mathsf{Term}(\mathcal{F}, \mathcal{V}, \mathcal{S})$ does not hold as higher-order variables are not in canonical form. The function $\uparrow$ takes a variable and returns the corresponding term $x\uparrow = \overline{y}_n.x(\overline{y}_n\uparrow)$ where $x : \overline{\sigma}_n \to a$, $\overline{y}_n : \overline{\sigma}_n$ and $x \notin \{\overline{y}_n\}$.

We refer to strings over $\mathbb{N}$ as *positions* (p, q, r). The empty string, denoted by ϵ, represents the *root position*. We write pq for the concatenation of two positions p and q and $p \leqslant q$ if p is a prefix of q, i.e., there exists a position r such that $pr = q$. In that case, we say that p is *above* q and q is *below* p. If neither $p \leqslant q$ nor $q \leqslant p$ then p and q are *parallel*, written $p \parallel q$. The set $\mathcal{P}\mathsf{os}(s)$ of positions of a term s is defined inductively as follows: $\mathcal{P}\mathsf{os}(\overline{x}_k.h(\overline{s}_n)) = \{\epsilon\} \cup \{ip \mid 1 \leqslant i \leqslant n \text{ and } p \in \mathcal{P}\mathsf{os}(s_i)\}$. Given a position $p \in \mathcal{P}\mathsf{os}(s)$, $s|_p$ denotes the *subterm* at position p where $s|_\epsilon = s$ and $(\overline{x}_k.h(\overline{s}_n))|_{ip} = \overline{x}_k.s_i|_p$. In particular, the scope of bound variables is not discarded, so the definition of subterms is well-defined in the presence of α-renaming. If $s|_p = \overline{x}_n.t$ we refer to $\overline{x}_n$ as $\mathsf{bv}(s, p)$. Furthermore, we use $s \trianglerighteq t$ to denote that t is a subterm of s and define $s \triangleright t$ as $s \trianglerighteq t$ and $s \neq t$. A position $p \in \mathcal{P}\mathsf{os}(s)$ is called *functional* (denoted by $p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(s)$) if the head of $s|_p$ is a function symbol. Given $p \in \mathcal{P}\mathsf{os}(s)$ and a term $\overline{x}_n.t$ of the same type as $s|_p = \overline{x}_n.u$ we define $s[\overline{x}_n.t]_p$ as the result of replacing u by t in s.

Finite mappings from variables to terms of the same type are called *substitutions* (θ, γ, δ, μ). Given a substitution $\theta = \{\overline{x}_n \mapsto \overline{t}_n\}$, its *domain* and *image* are defined as $\mathcal{D}\mathsf{om}(\theta) = \{\overline{x}_n\}$ and $\mathcal{I}\mathsf{m}(\theta) = \{\overline{t}_n\}$, respectively. The *free variables* introduced by a substitution θ are defined as the set $\mathsf{FV}(\theta) = \bigcup\{\mathsf{FV}(t) \mid t \in \mathcal{I}\mathsf{m}(\theta)\}$. A set of variables X is *fresh* for a substitution θ if $(\mathcal{D}\mathsf{om}(\theta) \cup \mathsf{FV}(\theta)) \cap X = \varnothing$. Given a substitution θ and a set of variables V, we denote by $\theta|_V$ the restriction of θ to variables in V. The application of a substitution θ to a term t (written in postfix notation $t\theta$) is defined hereditarily by implicitly performing β-reduction [6,14]: $x(\overline{t}_n)\theta = u\{\overline{x}_n \mapsto \overline{t}_n\theta\}$ if $x \mapsto \overline{x}_n.u \in \theta$, $h(\overline{t}_n)\theta = h(\overline{t}_n\theta)$ if $h \notin \mathcal{D}\mathsf{om}(\theta)$, and $(\overline{x}_n.t)\theta = \overline{y}_n.t\{\overline{x}_n \mapsto \overline{y}_n\}\theta$ where $\{\overline{y}_n\}$ is fresh for θ. Hence, $t\theta$ represents the capture-avoiding application of θ to the free variables of t. Given $\theta = \{\overline{x}_n \mapsto \overline{s}_n\}$ and $\delta = \{\overline{y}_m \mapsto \overline{t}_m\}$, we define their composition as $\theta\delta = \{\overline{x}_n \mapsto \overline{s}_n\delta\} \cup \{y_j \mapsto t_j \mid 1 \leqslant j \leqslant m \text{ and } y_j \notin \{\overline{x}_n\}\}$. A term s *matches* a term t ($s \leqslant_\beta t$) if there exists a substitution θ such that $s\theta = t$. In that case, t is called an *instance* of s. Moreover, if $s\theta = t\theta$ then θ is a *unifier* of s and t. Let W be a set of variables. Two substitutions θ_1 and θ_2 are *equal over* W ($\theta_1 = \theta_2$ $[W]$) if $\theta_1(x) = \theta_2(x)$ for all $x \in W$. A substitution θ is *at least as general* as a substitution δ with respect to W, denoted by $\theta \leqslant_\beta \delta$ $[W]$, if there exists a substitution γ such that $\theta\gamma = \delta$ $[W]$. We drop the suffix $[W]$ if W is the set of all variables. Two substitutions θ_1 and θ_2 are *orthogonal* ($\theta_1 \perp \theta_2$) if there is no substitution γ such that $\theta_1 \leqslant_\beta \gamma$ and $\theta_2 \leqslant_\beta \gamma$. Finally, given a binary relation R, $\theta_1\ R\ \theta_2$ denotes $\theta_1(x)\ R\ \theta_2(x)$ for all $x \in \mathcal{V}$.

A *variable renaming* is a bijective function from $\mathcal{V}$ to $\mathcal{V}$. Given a term s, a sequence of variables $\overline{x}_n$ and a set of variables W we define the $\overline{x}_n$-lifter of s away from W to be the substitution $\{z \mapsto \overline{y}_m.\rho(z)(\overline{x}_n, \overline{y}_m) \mid z : \overline{\tau}_m \to a \in \mathsf{FV}(s)\}$ where ρ is a variable renaming such that $\mathcal{D}\mathsf{om}(\rho) = \mathsf{FV}(s)$, $\mathcal{I}\mathsf{m}(\rho) \cap (W \cup \{\overline{x}_n\}) = \varnothing$, $\rho(z) : (\overline{\sigma}_n, \overline{\tau}_m) \to a$, $\overline{y}_m : \overline{\tau}_m$ and $\overline{x}_n : \overline{\sigma}_n$. Lifters will be needed to transform terms in potential overlaps into a form which is amenable to analysis with higher-order unification [7].

A *rule* $\ell \to r$ consists of two terms of the same sort where $\mathsf{FV}(r) \subseteq \mathsf{FV}(\ell)$ and the head of ℓ is not a variable. A rule $\ell \to r$ is *left-linear* if ℓ is linear. Furthermore, two rules $\ell_1 \to r_1$ and $\ell_2 \to r_2$ are *variants* if there exist substitutions θ_1, θ_2 such that $\ell_1\theta_1 = \ell_2$, $r_1\theta_1 = r_2$, $\ell_2\theta_2 = \ell_1$ and $r_2\theta_2 = r_1$. A (left-linear) *higher-order rewrite system* (HRS) is a set of (left-linear) rules. Given an HRS $\mathcal{R}$, its *rewrite relation* $\to_{\mathcal{R}}$ is defined as follows: There is a *rewrite step* $s \to_{\mathcal{R}} t$ if there exist a rule $\ell \approx r \in \mathcal{R}$, a substitution θ and a position $p \in \mathcal{P}\mathsf{os}(s)$ such that $s|_p = \overline{x}_n.\ell\theta$ and $t = s[\overline{x}_n.r\theta]_p$. In that case, we refer to s as a *redex*. Sometimes, we make the position p (as well as the rule $\ell \to r$ and substitution θ) explicit by writing $s \to^p_{\mathcal{R}} t$ ($s \to^{p|\ell\to r|\theta}_{\mathcal{R}} t$). Finally, we say that an HRS is *confluent* if ${}^*_{\mathcal{R}}\!\leftarrow \cdot \to^*_{\mathcal{R}} \subseteq \to^*_{\mathcal{R}} \cdot {}^*_{\mathcal{R}}\!\leftarrow$.

3 Deterministic Higher-Order Pattern Rewrite Systems

We now give a definition of deterministic higher-order patterns [15] for our notion of terms. To that end, we introduce the notion of expanded terms to represent the simply-typed λ-terms whose η-normal form is not an abstraction. For these terms, the expanded subterm relation corresponds to the higher-order subterm relation.

Definition 1. A term $\overline{x}_n.s$ is *expanded* if $\overline{x}_n.s = \overline{x}_n, \overline{y}_k.h(\overline{s}_m, \overline{y}_k{\uparrow})$ such that $\big(\{h\} \cup \mathsf{FV}(\overline{s}_m)\big) \cap \{\overline{y}_k\} = \varnothing$. Consider a term $\overline{x}_n.s$ as well as an expanded term $\overline{x}_n.t = \overline{x}_n, \overline{y}_k.h(\overline{t}_m, \overline{y}_k{\uparrow})$. We say that $\overline{x}_n.t$ is an *expanded subterm* of $\overline{x}_n.s$ ($\overline{x}_n.s \trianglerighteq_{\mathsf{e}} \overline{x}_n.t$) if there exist $n' \geqslant n$ and k terms written as $\overline{x}_{n'}.t_{m+1}, \ldots, \overline{x}_{n'}.t_{m+k}$ such that $\overline{x}_n.s \trianglerighteq \overline{x}_{n'}.h(\overline{t}_{m+k})$. We write $\overline{x}_n.s \triangleright_{\mathsf{e}} \overline{x}_n.t$ if $\overline{x}_n.s \trianglerighteq_{\mathsf{e}} \overline{x}_n.t$ and $\overline{x}_n.s \neq \overline{x}_n.t$.

Definition 2. A term s is a *deterministic higher-order pattern* (DHP) if the following conditions hold for all subterms $\overline{y}_n.x(\overline{t}_m)$ with $x \notin \{\overline{y}_n\}$ and $1 \leqslant i \leqslant m$: $\varnothing \neq \mathsf{FV}(t_i) \subseteq \{\overline{y}_n\}$, $\overline{y}_n.t_i$ is an expanded term, and $\overline{y}_n.t_i \not\trianglerighteq_{\mathsf{e}} \overline{y}_n.t_j$ whenever $i \neq j$. We refer to the terms $\overline{t}_m$ as *anchors* of the DHP. A *deterministic higher-order pattern rewrite rule* is a rewrite rule whose left-hand side is a DHP. A *deterministic higher-order pattern rewrite system* (DPRS) is a set of deterministic higher-order pattern rewrite rules.

Example 1. Consider the sort a, the function symbols $\mathsf{f} : \mathsf{a} \to \mathsf{a}$, $\mathsf{c} : \mathsf{a}$ as well as the variables $F : (\mathsf{a}, \mathsf{a}) \to \mathsf{a}$ and $G : (\mathsf{a}, \mathsf{a} \to \mathsf{a}) \to \mathsf{a}$. The terms $x, y.F(y, x)$, $x, y.F(\mathsf{f}(x), \mathsf{f}(y))$ and $x, y.F(x(y), x(\mathsf{c}))$ are DHPs. Note that only the first term also belongs to Miller's class of patterns [8]. On the other hand, $x, y.F(\mathsf{c}, x)$, $x, y.G(y, z.x(z, \mathsf{c}))$, $x, y.F(x, x)$, $x, y.F(\mathsf{f}(x), x)$ and $x.G(x(y), z.x(z))$ are not DHPs.

4 Critical Pairs

The crucial ingredient of any definition of critical pairs is a corresponding unification procedure. In [10], we present a sound and complete unification procedure for DHPs which produces minimal complete sets of unifiers.

Definition 3. A *minimal complete set of unifiers* U of two DHPs s and t satisfies the following conditions: $\mathcal{D}\mathsf{om}(\theta) \subseteq \mathsf{FV}(s) \cup \mathsf{FV}(t)$ for all $\theta \in U$, $s\theta = t\theta$ for all $\theta \in U$, if $s\delta = t\delta$ then $\theta \leqslant_\beta \delta\ [\mathsf{FV}(s) \cup \mathsf{FV}(t)]$ for some $\theta \in U$, if $\theta_1, \theta_2 \in U$ and $\theta_1 \neq \theta_2$ then $\theta_1 \perp \theta_2$.

The following definition of overlaps considers both function and variable positions but is designed such that it produces at most one critical peak per unifier in the minimal complete set of unifiers.

Definition 4. An *overlap* of a DPRS $\mathcal{R}$ is an octuple $\langle \ell_1 \to r_1, p, q, \overline{x}_n, \delta, \gamma, U, \ell_2 \to r_2 \rangle$ satisfying the following properties:

(i) $\ell_1 \to r_1, \ell_2 \to r_2 \in \mathcal{R}$,

(ii) $p \in \mathcal{P}\mathsf{os}(\ell_2)$, $\mathsf{bv}(\ell_2, p) = \overline{x}_n$ and δ is an $\overline{x}_n$-lifter of ℓ_1 away from $\mathsf{FV}(\ell_2)$,

(iii) $U' \neq \varnothing$ is a minimal complete set of unifiers of $\overline{x}_n.\ell_1\delta$ and $\ell_2\gamma|_{pq}$,

(iv) either (a) $q = \epsilon$, $\gamma = \varnothing$, $p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(\ell_2)$ and $U = U'$, or (b) $q = 1$ and

- $\ell_2|_p = \overline{x}_n.y(\overline{s}_m)$ where $y \notin \{\overline{x}_n\}$,
- $\gamma = \{y \mapsto \overline{y}_m.y''(y'(\overline{y}_m{\uparrow}), \overline{y}_m{\uparrow})\}$ with $y' : \overline{\sigma}_m \to b$ and $y'' : (b, \overline{\sigma}_m) \to a$ fresh variables assuming $y : \overline{\sigma}_m \to a$ and $\ell_1 : b$,
- $U = U' \setminus U''$ such that U'' is the set of all $\theta \in U'$ for which at least one of the following holds:
  - $\theta(y') = \overline{z}_m.z_i{\uparrow}$ for some $1 \leqslant i \leqslant m$
  - $\overline{z}_m.\ell_1 \leqslant_\beta \theta(y')$ whenever $\{\overline{z}_m\} \cap \mathsf{FV}(\ell_1) = \varnothing$,

(v) if $p = \epsilon$ then $\ell_1 \to r_1$ and $\ell_2 \to r_2$ are not variants.

For each $\theta \in U$ we obtain the *critical peak*

$$\ell_2\gamma\theta[(\overline{x}_n.r_1\delta)\theta]_{pq} \ {}^{pq|\ell_1\to r_1}_{\mathcal{R}}\!\leftarrow \ell_2\gamma\theta[(\overline{x}_n.\ell_1\delta)\theta]_{pq} = \ell_2\gamma\theta \to^{\epsilon|\ell_2\to r_2}_{\mathcal{R}} r_2\gamma\theta$$

which gives rise to the *critical pair* $\ell_2\gamma\theta[(\overline{x}_n.r_1\delta)\theta]_{pq} \approx r_2\gamma\theta$ of $\mathcal{R}$. The set of critical pairs of $\mathcal{R}$ is denoted by $\mathsf{CP}(\mathcal{R})$.

We close this section with an important result on the existence of unifiers:

Lemma 1. *Consider the terms s and t as well as the position $p \in \mathcal{P}\mathsf{os}(t)$. Let $\mathsf{bv}(t, p) = \overline{x}_n$ and δ be an $\overline{x}_n$-lifter of s away from $\mathsf{FV}(t)$. If there exist substitutions θ_1, θ_2 such that $\overline{x}_n.s\theta_1 = t|_p\theta_2$ then $\overline{x}_n.s\delta$ and $t|_p$ are unifiable.*

5 Orthogonal DPRSs are Confluent

Our definition of orthogonal DPRSs corresponds to the standard syntactic definition via left-linearity and absence of critical pairs [1, 7].

Definition 5. A DPRS $\mathcal{R}$ is *non-ambiguous* if $\mathsf{CP}(\mathcal{R}) = \varnothing$. We call a DPRS *orthogonal* if it is both left-linear and non-ambiguous.

We prove confluence of orthogonal DPRSs by establishing the diamond property ($\leftarrow \cdot \rightarrow \subseteq \rightarrow \cdot \leftarrow$) of their multi-step relation which lies between one-step and many-step rewriting. This general approach is often used to obtain confluence of orthogonal rewrite systems [7, 12].

Definition 6. Let $\mathcal{R}$ be an HRS. We define the *multi-step* relation $\multimap_{\mathcal{R}}$ inductively as follows:

(i) $\overline{x}_k.h(\overline{s}_n) \multimap_{\mathcal{R}} \overline{x}_k.h(\overline{t}_n)$ if $\overline{x}_k.s_i \multimap_{\mathcal{R}} \overline{x}_k.t_i$ for all $1 \leqslant i \leqslant n$, and

(ii) $\overline{x}_k.\ell\theta \multimap_{\mathcal{R}} \overline{x}_k.r\delta$ if $\ell \to r \in \mathcal{R}$, $\mathcal{D}\mathsf{om}(\theta) \subseteq \mathsf{FV}(\ell)$ and $\theta \multimap_{\mathcal{R}} \delta$.

This inductive definition of multi-steps is based on [5]. Even though Mayr and Nipkow use a different presentation of multi-steps, the following result can be obtained by adapting their proof accordingly [7].

Lemma 2. *For every* HRS $\mathcal{R}$, $\to_{\mathcal{R}} \subseteq \multimap_{\mathcal{R}} \subseteq \to_{\mathcal{R}}^*$.

For orthogonal DPRSs $\mathcal{R}$, we will obtain the diamond property for $\multimap_{\mathcal{R}}$ via the triangle property [12] of $\multimap_{\mathcal{R}}$ with respect to the following restriction of $\multimap_{\mathcal{R}}$.

Definition 7. Let $\mathcal{R}$ be an HRS. We define the *maximal multi-step* relation $\multimap\!\!\bullet_{\mathcal{R}}$ inductively as follows:

(i) $\overline{x}_k.h(\overline{s}_n) \multimap\!\!\bullet_{\mathcal{R}} \overline{x}_k.h(\overline{t}_n)$ if $\overline{x}_k.s_i \multimap\!\!\bullet_{\mathcal{R}} \overline{x}_k.t_i$ for all $1 \leqslant i \leqslant n$ and $h(\overline{s}_n)$ is not a redex, and

(ii) $\overline{x}_k.\ell\theta \multimap\!\!\bullet_{\mathcal{R}} \overline{x}_k.r\delta$ if $\ell \to r \in \mathcal{R}$, $\mathcal{D}\mathsf{om}(\theta) \subseteq \mathsf{FV}(\ell)$ and $\theta \multimap\!\!\bullet_{\mathcal{R}} \delta$.

We now proceed by establishing *coherence*. In the following, we say that an element $a \in A$ is *terminal* in a binary relation $R \subseteq A \times A$ if $a\ R\ b$ implies $b = a$. First, we show coherence of terminal anchors in redexes: A step with $\multimap$ starting from a term containing terminal anchors can also be performed when one abstracts away from these terminal anchors.

Lemma 3. *Let $\mathcal{R}$ be an orthogonal* DPRS, *$\ell \rhd \overline{x}_k.F(\overline{s}_n)$ for some $\ell \to r \in \mathcal{R}$ and let $\overline{x}_k.s_i$ be terminal in $\multimap_{\mathcal{R}}$ for all $1 \leqslant i \leqslant n$. If $\overline{x}_k.u\{\overline{y}_n \mapsto \overline{s}_n\} \multimap_{\mathcal{R}} \overline{x}_k.t$ then there is a term $\overline{y}_n.v$ such that $\overline{y}_n.u \multimap_{\mathcal{R}} \overline{y}_n.v$ and $\overline{x}_k.t = \overline{x}_k.v\{\overline{y}_n \mapsto \overline{s}_n\}$.*

In the proof of Lemma 3, we exploit the following: Since $\mathcal{R}$ is non-ambiguous, there cannot be an overlap between $\overline{x}_k.F(\overline{s}_n)$ and the left-hand side used in the multi-step $\overline{x}_k.u\{\overline{y}_n \mapsto \overline{s}_n\} \multimap_{\mathcal{R}} \overline{x}_k.t$. Hence, even if $\overline{x}_k.F(\overline{s}_n)$ and the left-hand side are unifiable, all their unifiers satisfy at least one of the conditions given for the set U'' in Definition 4. The former case is easily resolved since anchors are terminal. In the latter case, the multi-step can be performed without altering the anchors by definition. We can now prove the general case of coherence for orthogonal DPRSs.

Lemma 4. *Let $\mathcal{R}$ be an orthogonal* DPRS, *$\ell \to r \in \mathcal{R}$ and $\ell \rhd s$. If $s\theta \multimap_{\mathcal{R}} t$ then there exists a substitution δ such that $s\delta = t$ and $\theta|_{\mathsf{FV}(s)} \multimap_{\mathcal{R}} \delta$.*

Proof. By assumption, $s = \overline{x}_k.h(\overline{s}_n)$ is a linear term. Furthermore, $\{\overline{x}_k\}$ is fresh for θ without loss of generality. We perform induction on s and distinguish two cases.

- Assume $h \notin \mathcal{D}\mathsf{om}(\theta)$. If $s\theta \multimap_{\mathcal{R}} t$ (i) then $t = \overline{x}_k.h(\overline{t}_n)$ and $(\overline{x}_k.s_i)\theta \multimap_{\mathcal{R}} \overline{x}_k.t_i$ for all $1 \leqslant i \leqslant n$. By the induction hypothesis, there exist substitutions $\overline{\delta}_n$ such that $\overline{x}_k.t_i = (\overline{x}_k.s_i)\delta_i$ and $\theta|_{\mathsf{FV}(\overline{x}_k.s_i)} \multimap_{\mathcal{R}} \delta_i$ for all $1 \leqslant i \leqslant n$. Without loss of generality, $\{\overline{x}_k\}$ is fresh for δ_i for all $1 \leqslant i \leqslant n$. Since s is linear, the substitution $\delta = \bigcup_{i=1}^{n} \delta_i$ is well-defined and satisfies $\theta|_{\mathsf{FV}(s)} \multimap_{\mathcal{R}} \delta$. Finally, $s\delta = \overline{x}_k.h(\overline{s}_n\delta) = \overline{x}_k.h(\overline{t}_n) = t$. Now

suppose $s\theta \mathrel{-\!\!\circ\!\!\to_{\mathcal{R}}} t$ (ii), so there exist a rewrite rule $\ell' \to r' \in \mathcal{R}$ and a substitution θ' such that $s\theta = \overline{x}_k.\ell'\theta'$. In particular, $h \in \mathcal{F}$. Let p be the position such that $\ell|_p = s$. By Lemma 1, $\overline{x}_n.\ell'\mu$ and s are unifiable where μ is a $\overline{x}_k$-lifter of ℓ' away from $\mathsf{FV}(\ell)$. Let U be a complete set of unifiers of $\overline{x}_n.\ell'\mu$ and s. Then, $\langle \ell' \to r', p, \epsilon, \overline{x}_k, \mu, \varnothing, U, \ell \to r \rangle$ is an overlap in $\mathcal{R}$ which contradicts the fact that orthogonal DPRSs are non-ambiguous.

- Let $\theta(h) = \overline{y}_n.u = \overline{y}_n.h_u(\overline{u}_m)$, so $s\theta = \overline{x}_k.u\{\overline{y}_n \mapsto \overline{s}_n\}$. In particular, $\mathsf{FV}(s) = \{h\}$ as ℓ and therefore s is a DHP. Thus, the induction hypothesis yields that $\overline{x}_k.s_i$ is terminal in $\mathrel{-\!\!\circ\!\!\to_{\mathcal{R}}}$ for all $1 \leqslant i \leqslant n$. From Lemma 3, we obtain a term $\overline{y}_n.v$ such that $\overline{y}_n.u \mathrel{-\!\!\circ\!\!\to_{\mathcal{R}}} \overline{y}_n.v$ and $t = \overline{x}_k.v\{\overline{y}_n \mapsto \overline{s}_n\}$. Hence, we can define $\delta = \{h \mapsto \overline{y}_n.v\}$ to obtain $s\delta = \overline{x}_k.v\{\overline{y}_n \mapsto \overline{s}_n\} = t$ and $\theta(h) \mathrel{-\!\!\circ\!\!\to_{\mathcal{R}}} \delta(h)$ as desired. □

Observe that in the proof of Lemma 4, the fact that anchors are terminal is a direct consequence of the induction hypothesis. This assumption greatly simplifies the proof of Lemma 3. Having established coherence for orthogonal DPRSs, we can now just replay the proof of the triangle property for PRSs adapted to our presentation of multi-steps [7].

Lemma 5. *For every orthogonal* DPRS $\mathcal{R}$*, if* $s \mathrel{-\!\!\circ\!\!\to_{\mathcal{R}}} t$ *then* $t \mathrel{-\!\!\circ\!\!\to_{\mathcal{R}}} u$ *for the term* u *that satisfies* $s \mathrel{-\!\!\bullet\!\!\to_{\mathcal{R}}} u$*.*

The diamond property is a direct consequence of the triangle property [12].

Corollary 1. *Multi-step rewriting has the diamond property for orthogonal* DPRSs*.*

Theorem 1. *Orthogonal* DPRSs *are confluent.*

Proof. Let $\mathcal{R}$ be an orthogonal DPRS. Since $\mathrel{-\!\!\circ\!\!\to_{\mathcal{R}}}$ has the diamond property (Corollary 1) and $\to \subseteq \mathrel{-\!\!\circ\!\!\to_{\mathcal{R}}} \subseteq \to^*$ (Lemma 2), we conclude confluence of $\mathcal{R}$ by standard results for abstract rewriting, e.g. [1, Corollary 2.7.7]. □

Example 2. The left-linear DPRS $\mathcal{R}$ consisting of the rules

$$\begin{aligned}
\mathsf{sum}(\mathsf{nil}) &\to \mathsf{0}\\
\mathsf{sum}(\mathsf{cons}(x, xs)) &\to x + \mathsf{sum}(xs)\\
\mathsf{comp}(z_3.\mathsf{sum}(z_3), z_0.\mathsf{foldr}_1(z_1, z_2.G(z_1, z_2), xs, z_0), ys) &\to \mathsf{check}(z_1, z_2.\mathsf{sum}(G(z_1, z_2)), xs, ys)\\
\mathsf{check}(z_1, z_2.F(z_1, \mathsf{sum}(z_2)), xs, ys) &\to \mathsf{foldr}_2(z_1, z_2.F(z_1, z_2), \mathsf{sum}(xs), ys)\\
\mathsf{from}(x) &\to \mathsf{cons}(x, \mathsf{from}(s(x)))
\end{aligned}$$

models a basic fusion transformation rule for folds in functional programming together with a rule to construct infinite lists $[i, i+1, \dots]$. Here, we denote function symbols by sans-serif font and omit type information for brevity. In particular, $\mathcal{R}$ uses the expressivity of DHPs in order to neatly model transformations of compositions of a fold and the sum function to just one fold combining the two operations:

$$\begin{aligned}
& z_0.\mathsf{comp}(z_1.\mathsf{sum}(z_1), z_1.\mathsf{foldr}_1(z_2, z_3.\mathsf{cons}(z_2 \times z_2, z_3), \mathsf{nil}, z_1), z_0)\\
\to_{\mathcal{R}}\ & z_0.\mathsf{check}(z_1, z_2.\mathsf{sum}(\mathsf{cons}(z_1 \times z_1, z_2)), \mathsf{nil}, z_0)\\
\to_{\mathcal{R}}\ & z_0.\mathsf{check}(z_1, z_2.z_1 \times z_1 + \mathsf{sum}(z_2)), \mathsf{nil}, z_0)\\
\to_{\mathcal{R}}\ & z_0.\mathsf{foldr}_2(z_1, z_2.z_1 \times z_1 + z_2, \mathsf{sum}(\mathsf{nil}), z_0)\\
\to_{\mathcal{R}}\ & z_0.\mathsf{foldr}_2(z_1, z_2.z_1 \times z_1 + z_2, \mathsf{0}, z_0)
\end{aligned}$$

Clearly, $\mathcal{R}$ is not terminating. However, it is non-ambiguous, so $\mathcal{R}$ is orthogonal and therefore confluent by Theorem 1.

Acknowledgments. We would like to thank the anonymous reviewers for their valuable and extensive feedback.

References

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

[2] Henk P. Barendregt. Lambda calculi with types. In Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press, Oxford, UK, 1992. doi:10.1093/oso/9780198537618.003.0002.

[3] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, 1940. doi:10.2307/2266170.

[4] Thiago Felicissimo and Jean-Pierre Jouannaud. Sort-based confluence criteria for non-left-linear higher-order rewriting. In *Proc. 30th International Conference on Automated Deduction*, pages 226–245, Cham, 2025. Springer. doi:10.1007/978-3-031-99984-0_13.

[5] Nao Hirokawa and Aart Middeldorp. Decreasing diagrams and relative termination. *Journal of Automated Reasoning*, 47(4):481–501, 2011. doi:10.1007/s10817-011-9238-x.

[6] Stefan Kahrs. Context rewriting. In Michaël Rusinowitch and Jean-Luc Rémy, editors, *Proc. 3rd International Workshop on Conditional Term Rewriting Systems*, volume 656 of *Lecture Notes in Computer Science*, pages 21–35, 1992. doi:10.1007/3-540-56393-8_2.

[7] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, 1998. doi:10.1016/S0304-3975(97)00143-6.

[8] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991. doi:10.1093/logcom/1.4.497.

[9] Johannes Niederhauser and Aart Middeldorp. Towards confluence of deterministic higher-order pattern rewrite systems by critical pairs. In Raúl Gutiérrez and Naoki Nishida, editors, *Proc. 14th International Workshop on Confluence*, pages 12–18, 2025.

[10] Johannes Niederhauser and Aart Middeldorp. Unification of deterministic higher-order patterns (full version). *CoRR*, abs/2601.14211, 2026. Accepted at IJCAR 2026. doi:10.48550/arXiv.2601.14211.

[11] Tobias Nipkow. Higher-order critical pairs. In *Proc. 6th Annual Symposium on Logic in Computer Science*, pages 342–349, New York, NY, 1991. IEEE Computer Society. doi:10.1109/LICS.1991.151658.

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

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

[14] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework: The propositional fragment. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Proc. 3rd International Workshop on Types for Proofs and Programs*, volume 3085 of *Lecture Notes in Computer Science*, pages 355–377, 2004. doi:10.1007/978-3-540-24849-1_23.

[15] Tetsuo Yokoyama, Zhenjiang Hu, and Masato Takeichi. Deterministic higher-order patterns for program transformation. In Maurice Bruynooghe, editor, *Proc. 13th International Symposium on Logic-Based Program Synthesis and Transformation*, volume 3018 of *Lecture Notes in Computer Science*, pages 128–142, Berlin, Heidelberg, Germany, 2004. Springer. doi:10.1007/978-3-540-25938-1_12.

Confluence of bang modulo

Giulio Guerrieri and Vincent van Oostrom

University of Sussex, Falmer, United Kingdom
G.Guerrieri@sussex.ac.uk and Vincent.van-Oostrom@sussex.ac.uk

Abstract

We show that for the (untyped) bang-calculus $\beta!$ is confluent modulo σ.

Introduction. The bang-calculus [8, 10, 11] is an *untyped* version of the implicative fragment of the call-by-push-value calculus [16], which is *typed*. The calculi share that they subsume the call-by-name and call-by-value paradigms. For definiteness we recall the calculus studied here in Tab. 1. However, so as to be able to *instantiate* higher-order term rewriting theory [18, 25], we will exclusively employ its presentation as a higher-order pattern rewrite system[1] (PRS). The simply-typed signature of the PRS presentation has a base-type term and function symbols:

$$\mathsf{abs} \;:\; (\mathsf{term} \to \mathsf{term}) \to \mathsf{term} \qquad \mathsf{app} \;:\; \mathsf{term} \to \mathsf{term} \to \mathsf{term} \qquad \mathsf{bang} \;:\; \mathsf{term} \to \mathsf{term}$$

It extends the PRS signature of the (untyped) λ-calculus [18, Ex. 3.4][25, Ex. 11.2.6(i)] with the bang. We ad(a/o)pt standard notational conventions of the λ-calculus [4, 25] abbreviating $\mathsf{abs}\,(x.M)$ to $\lambda x.M$, $\mathsf{app}\,M\,N$ to $M\,N$ and $\mathsf{bang}\,M$ to $!M$. Its $\beta!$- and σ-rewrite rules then read:

$$\begin{array}{llll} \beta! & : & (\lambda x.S(x))\,!T \to S(T) & \qquad \sigma_1 \;:\; (\lambda x.S(x))\,T\,V \to (\lambda x.S(x)\,V)\,T \\ \sigma_2 & : & (\lambda x\,y.S'(x,y))\,T \to \lambda y.(\lambda x.S'(x,y))\,T & \qquad \sigma_3 \;:\; V\,((\lambda x.S(x))\,T) \to (\lambda x.V\,S(x))\,T \end{array}$$

Remark. Tab. 1 employs a *grammar* instead of a *signature*, *rule schemata* instead of *rules*, and deals with *renaming* and *substitution* explicitly (it has to) instead of absorbing those, in PRSs. We will heavily rely on that PRSs are preserved under various syntactic transformations such as coloring (see below), so we still have PRS theory [18, 25] and PRS tools[1] at our avail. This is to be contrasted to the literature on the λ-calculus using such, invariably rife with well bad *redundancy* or *handwaving*; it must be as the λ-calculus is *not* closed under coloring (残念).

The bang reflects the !-type from linear logic, denoting that resources are *replicable*, in the syntax. In particular, the $\beta!$-rule makes explicit that arguments to functions should be replicable.

Example 1. For $\omega := \lambda x.x\,!x$, the bang-calculus version of the *mockingbird* [23][4, Def. 6.2.1], we have the infinite reduction $\omega\,((\lambda x.!\omega)\,(y\,z)) \to_{\sigma_3} Z \to_{\beta!} Z \to_{\beta!} \dots$ for $Z := (\lambda x.\omega\,!\omega)\,(y\,z)$.

Here we resolve the problem whether $\beta!$ is confluent modulo σ, a problem dating back to [8, 11], in the affirmative. After making both the problem and its resolution formal statements, we investigate the rewrite properties of $\beta!$ and σ and find the latter and its combination with the former to be non-trivial. Based on that analysis we introduce the $B!$-*rule* (B is the Greek capital β, pronounced here as *big beta*) obtained by *saturating* the redex-pattern of the $\beta!$-rule with *Dyck-spines*, in the spirit of [6]. A Dyck-spine (Def. 1) is a context where the applications and abstractions along its head-spine [5, Def. 4.2] match up as the opening and closing parentheses of the Dyck language $\mathcal{D}$ of matching parentheses, see, *e.g.*, [22], with the hole at its coccyx / tip. Dyck-spines are designed to be closed under σ-equivalence, enabling to speak of *residuals* [7, 19] of $B!$-redex-patterns *after* both (other) $B!$- and σ-steps, thereby enabling resolution of the problem via Church & Rosser's classical method of *confluence-by-orthogonality* [4, 18, 3, 25], but for $B!$ instead of for $\beta!$. A flattening argument allows us to conclude.

[1] The PRS is downloadable from http://www.javakade.nl/research/trs/bang.trs in the HRS format of the Confluence Competition, supported by tools such as Nagele's CSI^ho and Hamana's SOL.

$$!\Lambda \quad \ni \quad s,t ::= x \mid \lambda x.t \mid s\,t \mid !t$$

$$\beta! : (\lambda x.t)\,!s \to t\{s/x\} \qquad \sigma_1 : (\lambda x.s)\,t\,v \to (\lambda x.s\,v)\,t \;\text{ if } x \notin \mathsf{fv}(v)$$
$$\sigma_2 : (\lambda x.\lambda y.s)\,t \to \lambda y.(\lambda x.s)\,t \;\text{ if } y \notin \mathsf{fv}(t) \qquad \sigma_3 : v\,((\lambda x.s)\,t) \to ((\lambda x.v\,s)\,t) \;\text{ if } x \notin \mathsf{fv}(v)$$

Table 1: Bang-calculus presentation [8, 10, 11] by *grammar* and *rule schemata*, **not** used here

Remark. The idea underlying $B!$ of *saturating* left-hand sides (here of the $\beta!$-rule) under *administrative* rules (here σ-rules) appears in many guises in the literature, see *e.g.*, [6, §4.3][21, 2, 15, 25]. It was later vulgarised by Accattoli and Kesner as *reduction-at-a-distance* for the simple (*regular* not *context-free*) case of a λ-calculus with explicit substitutions *à la* De Bruijn [1].

Preliminaries. We assume familiarity with rewriting [3, 25]. We employ $\to$, $\twoheadrightarrow$, and $\sim$ (each possibly subscripted) to denote respectively the rewrite system (relation) induced by the $\beta!$-rule, its *finite* reductions (reducibility), and conversions (convertibility) induced by the σ-rules with $\sigma := \{\sigma_1, \sigma_2, \sigma_3\}$, and use $[a]$ to denote a *modulo* $\sim$ defined by $\{b \mid a \sim b\}$, and $[\to]$ to denote $\to$ *modulo* $\sim$ defined on such equivalence classes by $[a]\;[\to]\;[b]$ if $a \sim \cdot \to \cdot \sim b$. Our problem resolution (Cor. 1) asserts that $\beta!$ is *confluent modulo* $\sim$, *i.e.* $[\to]$ is confluent.

Remark. The notion *confluence of* $\to$ *modulo* $\sim$ is not unequivocally defined in the literature. It is oftentimes equivocated [13][25, Ex. 14.3.1(iii)] with confluence of $\sim \cdot \to \cdot \sim$, not equivalent to confluence of $[\to]$ as shown by $b \leftarrow a \to c$ and $b \sim c$, though only slightly so; they can be reconciled by asking for a common reduct for $\sim \cdot \to \cdot \sim$ *up to* $\sim$. It is also sometimes defined [12, p. 802] as $\twoheadleftarrow \cdot \sim \cdot \twoheadrightarrow \subseteq \twoheadrightarrow \cdot \sim \cdot \twoheadleftarrow$, not equivalent to the other two as shown by $a \sim b \to c$.

Below, we will show (Cor. 1) confluence of $[\to]$ via the *diamond* property (DP) of $\sim \cdot \multimap$ (Thm. 2), where $M \multimap N$ denotes there is a *complete* $\beta!$-*development* (CD) from M to N, a reduction from M in which only *residuals* [7, 19] of redex-patterns in M may be contracted with none remaining in N. Following [4, 25], we formalise both notions by *coloring*[4] occurrences of symbols, where *coloring* pairs a symbol up with a natural number, written as superscript / using color (since application is implicit: of its parentheses). The *colored* PRS rules are (Fig. 1):[2]

$$\beta!^i : ((\lambda^i x.S(x))^i(!^i T) \to S(T) \qquad \sigma_1 : ((\lambda^k x.S(x))^j T)^i V \to (\lambda^k x.(S(x))^i V)^j T$$
$$\sigma_2 : (\lambda^j x.\lambda^k y.S(x,y))^i T \to \lambda^k y.(\lambda^j x.S(x,y))^i T \qquad \sigma_3 : (V)^i((\lambda^k x.S(x))^j T) \to (\lambda^k x.(V)^i S(x))^j T$$

Example 2. If $M \to N$, then also $M \multimap N$ by a CD of its *singleton* redex-pattern occurrence indicated by coloring, *e.g.*, $(\lambda x.(\lambda x.x\;!x)\;!\omega)\;(y\;z)$ witnesses $Z \multimap Z$ for Z in Ex. 1. Coloring extends to multiple redex-pattern occurrences, *e.g.*, $(\lambda x.u\;((\lambda y.x\;y)\;!b))\;!a \multimap u\;(a\;b)$ is witnessed by $(\lambda x.u\;((\lambda y.x\;y)\;!b))\;!a \to_{\beta!} u\;((\lambda x.x\;b)\;!a) \to_{\beta!} u\;(a\;b)$ (starting with $\to_{\beta!}$ gives the same CD).

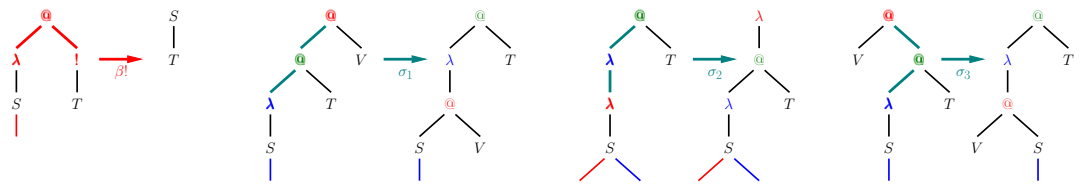


Figure 1: (Red) $\beta!$ and (polychrome) σ_i with redex-patterns [25, §8.6.2] in **bold**.

[2] A colored edge below a leaf S indicates that (the unique) λ (of that color) binds in terms substituted for S.

No(ta)tions extend naturally to coloring, *e.g.*, $\sim$ denotes $\sim_\sigma$ where σ denotes $\{\sigma_1, \sigma_2, \sigma_3\}$. Any map on colors, a *discoloring*, induces a morphism on terms, steps, (finite and infinite) reductions and conversions. *Matting* is the discoloring that maps all colors to *matte* (black), 0. As mapping terms to matte terms is an embedding, we may and will identify the former with the latter.

$\beta!$. Given a set $0 \notin \mathcal{C}$ of colors we use $\beta!$ and $\to$ to denote $\bigcup_{i\in\mathcal{C}} \beta!^i$ and $\to_{\beta!}$, respectively. A term M is a *$\mathcal{C}$-coloring* of M if only colors in $\mathcal{C} \cup \{0\}$ occur in M and colors in $\mathcal{C}$ *only* occur in $\beta!$-redex-patterns. A ($\beta!$-)*development* of M is a $\to$-reduction that is the matting of a $\to$-reduction from *some* $\mathcal{C}$-coloring M of M; if it ends in a $\to$-normal form with matting N, then $M \multimap\!\!\to N$ is *complete* (a CD). Note that M is matte if it is in $\to$-normal form.

Example 3. Non-red-colorings of Z in Ex. 1 are $(\lambda x.(\lambda x.x\,!x)\,!\omega)\,(y\,z)$ (non-monochrome), $(\lambda x.(\lambda x.x\,!x)\,!\omega)\,(y\,z)$ (blue) and $(\lambda x.(\lambda x.x\,!x)\,!\omega)\,((y)\,z)$ (non-redex-pattern). Ex. 2 exemplifies red,blue-coloring. $J \multimap\!\!\to I\,!I$ but not $J \multimap\!\!\to I$ for $I := \lambda x.x$, $J := I\,!I\,!I$ (1 redex-pattern in J).

Theorem 1. *$\multimap\!\!\to$ has the DP, because $\to$ is terminating on $\mathcal{C}$-colorings and preserves them.*

Proof. This holds for *orthogonal* PRSs [18] (OPRSs; *left-linear* and without *critical peaks*), with $\beta!$ orthogonal (on its own)[1]: $\mathcal{C}$-colorings are preserved by the contracted $\beta!$-redex-pattern not overlapping others. Termination follows from [25, Thm. 11.5.11]. Both CDs of $N \multimap\!\!\leftarrow M \multimap\!\!\to L$ lift to *developments* from some $\mathcal{C}$-coloring M of M for $\mathcal{C}$ the union of the colors for the CDs; completing them yields $M \multimap\!\!\to P \multimap\!\!\leftarrow L$ for P the (unique and matte) $\to$-normal form of M. □

The theory of orthogonality [25, Ch. 8] not just gives confluence of $\beta!$ [10] for free (by $\to \subseteq \multimap\!\!\to \subseteq \twoheadrightarrow$ [25, Prop. 1.1.11]), it allows to *construct a least upper bound* for any peak of $\beta!$-reductions.

σ. Since rewriting *modulo* equivalences classes is unwieldy, one may try to transform the σ-rules into a *complete* rewrite system [17, 20]. This is non-trivial for various reasons: (i) *orienting* σ into rewrite rules in *any* direction yields PRSs that, though terminating, have *non-joinable* critical peaks [18] as easily checked by tools[1], *e.g.* the one shown in Fig. 2 (left); (ii) Knuth–Bendix *completion* of the above σ-rules even leads to a *non-orientable* rule, with either side embedding the other, the *swap*-equivalence $\tau{:}(\lambda x.((\lambda y.S(y,x))\,T))\,V \to (\lambda y.((\lambda x.S(y,x))\,V))\,T$ depicted in Fig. 2 (right). (iii) turning the σ-rules into a *first-order* TRS by *erasing* bound variables, completion using tools fails[3] (completing σ should be at least as difficult); (iv) the *reverse* σ-rules are not well-behaved in the same way the η-rule [4] is not: they test for the *absence* of variables, i.e. perform a so-called *no-occur-check* [20, Rem. 3.4.23]. (v) whereas the σ-rules of [6, 21] give a *conservative* extension of β-convertibility [4], that fails for our σ-rules w.r.t. $\beta!$-convertibility (*intuitively* by ! being absent, *formally* by Fig. 2 and Cor. 1).

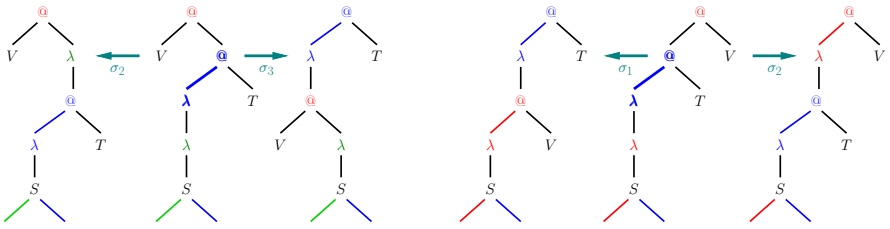


Figure 2: Non-joinable (left) and non-orientable (right) critical peaks between σ_2 and σ_3 / σ_1.

[3]The TRS in mkbTT-format is downloadable from `http://www.javakade.nl/research/trs/sigma.trs`.

$$
\begin{array}{lcrcll}
M_1 & := & (\lambda x.u\,((\lambda y.x\,y)\,!b))\,!a & \to_{\beta!} & u\,((\lambda x.x\,b)\,!a) & =: N \\
M_2 & := & u\,((\lambda x.(\lambda y.x\,y)\,!b))\,!a) & \to_{\beta!} & N & \\
M_3 & := & u\,(((\lambda x.\lambda y.x\,y)\,!a)\,!b) & \to_{B!} & N & \\
M_4 & := & u\,((\lambda y.(\lambda x.x\,y)\,!a)\,!b) & \to_{\beta!} & N & \\
 & & M_4 & \to_{\beta!} & u\,((\lambda y.a\,y)\,!b) & =: L
\end{array}
$$

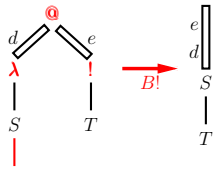


Figure 3: Exemplifying saturation of $\beta!$ into $B!$ (left) and presentation of (a red) $B!$ (right).

Open Problem. Is there a complete PRS presenting the equational theory $\sim$ of σ?

Observe that $\sim$ *is* decidable since $\sim$-classes are finite by σ being *size-preserving*. That does not resolve the open problem since decidable equational theories need not be presentable by complete (finite) term rewrite systems [14, 24], but it does make Cor. 1 below *effective*. In turn, size-preservation is a consequence of that the σ-rules only *reconfigure* symbols (as seen in Fig. 1; they are *chemical* in the sense of [25, Example 8.6.1]). That also makes that matting is (the converse of) a *labelling* in the sense of [25, Def. 8.4.5(i)] on σ-conversions; they lift uniquely:[4]

Lemma 1. *A $\sigma_i^{(-1)}$-step from a matting M of M lifts to a unique $\sigma_i^{(-1)}$-step from M per Fig. 1.*

Non-coherence of σ with $\beta!$. A major obstacle to confluence of $[\to]$ is that $\sim \cdot \to \subseteq \to \cdot \sim$ does *not* hold; σ need *not* cohere with $\beta!$; *e.g.*, it does not in Ex. 1. We present our technique to overcome it, of *saturating* $\beta!$ to *make* σ cohere with it, on the following running example.

Example 4. The terms M_1 and M_4 in Fig. 3 (left) are σ-convertible by $M_1 \sim_{\sigma_3^{-1}} M_2 \sim_{\sigma_1^{-1}} M_3 \sim_{\sigma_2} M_4$. Both M_1 and M_4 have *two* $\beta!$-redex-patterns, colored red and blue, but M_3 in the middle only has a *single* (blue) one; the @ and λ in it have become (temporarily) divorced. So, $M_3 \sim M_1 \to_{\beta!} N$, but for *no* P we have $M_3 \to_{\beta!} P \sim N$, and the same holds when discoloring.

Our aim is to make the red symbols in M_3 constitute a redex-pattern for a *generalised* $\beta!$-rule we dub $B!$. To that end, we *saturate* the left-hand side $(\lambda x.S(x))\,!T$ of the $\beta!$-rule by allowing for arbitrary so-called *Dyck-spines* d and e betwixt its @ and its λ respectively !, see Fig. 3 (right), where bars $[]$ represent Dyck-spines and the $B!$-redex-pattern is in **bold**, *cf.* $\beta!$ in Fig. 1.

Definition 1. A *Dyck-spine* is a single-hole context in $\mathcal{DS} \ni d, e ::= \Box \mid (d[\lambda^i x.\Box])^i M \mid d[e]$. We let $\vec{d}$ denote the sequence of variables bound by the λ-abstractions on the *spine* toward the coccyx / tip $\Box$ of d. The (colored) PRS $B!$-rule scheme for Dyck-spines d, e is, Fig. 3 (right):

$$B!^i \quad : \quad (d[\lambda^i x.S(x,\vec{d})])^i e[!^i T(\vec{e})] \to e[d[S(T(\vec{e}),\vec{d})]]$$

Intuitively [6, 2], Dyck-spines are *vertically* context-free; we think of @ and λ *along the spine toward* $\Box$ as *opening* and *closing* parentheses in the Dyck language of *matching* parentheses over the alphabet $\{(,)\}$. Indeed, given an occurrence of the @ of a $B!$-redex-pattern its corresponding λ is found by *matching* (using a PDA [22]) along the spine, and similarly for !. We extend our conventions: $B!$ denotes $\bigcup_{i\in\mathcal{C}} B!^i$ and M is a $\mathcal{C}$*-coloring* of M, if M only uses colors in $\mathcal{C} \cup \{0\}$ and colors in $\mathcal{C}$ exclusively occur in $B!$-redex-patterns. $\beta!$ is a *flat* $B!$ arising from $d, e := \Box$.

Example 5. In Fig. 3 (left), $M_3 = u\,((d[\lambda y.x\,y])\,e[!b]) \to_{B!} u\,(e[d[x\,b]]) = N$ for $d := (\lambda x.\Box)\,!a$ and $e := \Box$, regaining coherence. For the occurrence of @ in M_3 we find its corresponding λy

[4] $\beta!$-reductions need *not* lift, since the left-hand side of $\beta!$ being monochrome *restricts* it [25, Ex. 8.4.28]!

by repeatedly walking to the left subterm, first encountering a @ (pushing that), then a λx (popping the @), and finally the matching λy. Similarly, $(d[\lambda x.x\,!x])\,e[!\omega] \to_{B!} e[d[\omega\,!\omega]]$ for $d := \Box$ and $e := (\lambda x.\Box)\,(y\,z)$ in Ex. 1, again regaining coherence since $e[d[\omega\,!\omega]] = Z$.

Lemma 2. *$\mathcal{C}$-colorings are preserved by $\sim$. (Note: σ-steps are* a priori not *constrained by $\mathcal{C}$!)*

Proof. First, note that if a side of a σ-rule occurs in a $\mathcal{C}$-coloring M of M, then its occurrences of @ and its left child λ as in Fig. 1 both have the same color; either both matte (neither in a redex-pattern) or both in $\mathcal{C}$ (they must match). Next, we consider peaks between $B!$- and such σ-redex-patterns, where we assume w.l.o.g. both symbols are blue. The only interesting case is when there is overlap. Projecting onto their spine(s), the steps induced by such σ are:

$$(() \to_{\sigma_1} ()(\qquad ()) \to_{\sigma_2})() \qquad (\to_{\sigma_3} ()(\qquad () \to_{\sigma_3} \varepsilon$$

expressed using matching parentheses, where the 2 rules for σ_3 correspond to @,@ being on *distinct* spines in its left-hand side. The Dyck language $\mathcal{D}$ is closed under the (reverse) steps. □

Example 6. M_1 in Fig. 3 is red,blue-colored, so all M_i are by them being σ-convertible to M_1.

Coherence of σ with $B!$. We have σ is *coherent* with $B!$, as we aimed for.

Lemma 3. *$M' \to_{B!} N' \sim N$ for some $\mathcal{C}$-colored N', if $M' \sim M \to_{B!} N$ for $\mathcal{C}$-colored M', M, N.*

Proof. By transitivity of $\sim$ it suffices to show the inclusion for each *local cliff* $M' \leftrightarrow M \to_{B!} N$. (critical) If one of the (three) symbols in a side of a $\sigma_i^{(-1)}$-rule (Fig. 1) overlaps with the head-symbol @ of $B!$ or its matching λ (Fig. 3), the cliff is *critical* and handled as *per* Fig. 4. (orthogonal) Otherwise, the pattern of the σ-rule must either be *schematic*, on one of the *Dyck-spines* d or e of the lhs $d[\lambda x.S(x,\vec{d})]\,e[!T(\vec{e})]$ of the $B!$-rule, or in its *body* (obtained by instantiating S) or in the *argument* (obtained by instantiating T) as usual for OPRSs. A common reduct is found by contracting *residual* σ-redex-patterns in the rhs of $B!$ [7, 3, 25]. □

Remark. In Fig. 4 we have *distinctly* colored *non-matching* symbols, and overlapping symbols are in **bold**.[5] That the various σ-equivalences hold, uses that binders in Dyck-spines d and e do not bind in each other (per the variable convention), so that $d[e] \sim e[d]$. For instance, if $d = (\lambda y.\Box)\,y$ and $e = (\lambda z.\Box)\,z$, then $d[e[N]] = (\lambda y.(\lambda z.N)\,z)\,y \sim_\tau (\lambda z.(\lambda y.N)\,y)\,z = e[d[N]]$.

Example 7. Lem. 3 for the cliff $M_3 \sim M_1 \to_{B!} N$ in Fig. 3 (left) yields $M_3 \to_{B!} N \sim N$.

Flattening $B!$ to $\beta!$. The Dyck-language $\mathcal{D}$ has the following rewrite-characterisation:

$w \in \mathcal{D}$ iff the normal form of w for the rule $(() \to ()($ is *flat*, that is, of shape $()\ldots()$.

Flattening, σ_1, σ_3-reducing M to a[6] σ_1, σ_3-normal form, extends the idea from words to terms:

Proposition 1. *A flattening of a $\mathcal{C}$-coloring M of M is flat (all $B!$-redexes are $\beta!$-redexes).*

Proof. In a $B!$-redex $(d[\lambda x.L])\,e[!P]$ in a flattened term, both d, e are flattened entailing they are sequences of @–λ-pairs (σ_1 would be applicable to a minimal nesting). Then if d / e were non-empty, σ_1 / σ_3 would be applicable to the head-@ of $B!$ and the first @–λ-pair in d / e. □

Example 8. For M_1 and M_2 in Fig. 3 (left), flattening gives rise to[7] $M_1^\flat := (\lambda x.(\lambda y.u\,(x\,y))\,!b)\,!a$, and for M_3 and M_4 to $M_4^\flat := (\lambda y.(\lambda x.u\,(x\,y))\,!a)\,!b$, which indeed both are flat. (They are also swap-equivalent, but Fig. 2 (left) shows that in general that fails for σ-convertible terms.)

[5] Dyckness of d rules out the *third* way to overlap σ_1 with $B!$, *only* on the λ *not* on the @ of $B!$.

[6] σ_1, σ_3-reduction is terminating [8] (as tools also show) but not confluent (σ_1, σ_3 have non-joinable critical peaks, *e.g.*, in $(\lambda x.z)\,x\,((\lambda y.z)\,y)$). It can be shown that σ_1, σ_3-reduction is confluent modulo swap-equivalence τ.

[7] Beware: $M^\flat$ isn't notation for *the* flattening of M. We use $M^\flat$ to denote *a* flat term such that $M \sim M^\flat$.

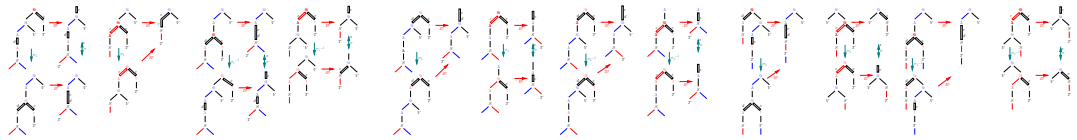


Figure 4: Joining critical peaks of $B!$ and the $\sigma_i^{(-1)}$ up to σ (zoom-in for extra legibility).

Coherence of σ with $\beta!$-developments

Lemma 4 (σ-transfer)**.** *If $M^\flat$ is flat*[7] *in $M^\flat \sim M \multimap N$, then $M^\flat \multimap N' \sim N$ for some N'.*

Proof. $M^\flat \sim M \twoheadrightarrow N$ for some $\mathcal{C}$-colorings M of M and $M^\flat$ of $M^\flat$ with $M^\flat \sim M$ by assumption, definition of $\multimap$ and Lem. 1,2. $M^\flat \twoheadrightarrow_{B!} N' \sim N$ for some colored N' by iterating Lem. 3. Since $M^\flat$ is flat, so is $M^\flat$, so all $B!$-redex-patterns in it are $\beta!$-redex-patterns. These are preserved by contracting other such (Thm. 1), so in fact $M^\flat \twoheadrightarrow_{\beta!} N' \sim N$ using σ retains color. □

Example 9. $M_1^\flat \sim M_4 \multimap u\,(a\,b)$ in Ex. 8, transfers to $M_1^\flat \multimap u\,(a\,b) \sim u\,(a\,b)$ by Lem. 4.

Theorem 2. $\multimap \cdot \sim \cdot \multimap \subseteq \sim \cdot \multimap \cdot \multimap \cdot \sim$ *($\sim \cdot \multimap$ has the* diamond *property).*

Proof. If $M \sim M_i \multimap N_i$ for $i \in \{1,2\}$, then $M^\flat \sim M_i \multimap N_i$ for $M^\flat$ a flattening of M. Lem. 4 transfers that to $M^\flat \multimap N_i' \sim N_i$ for some N_i'. By DP of $\multimap$ (Thm. 1) $N_i' \multimap L$ for an L. □

Example 10. Thm. 2 for $N \leftarrow M_1 \sim M_4 \rightarrow L$ in Fig. 3 gives $N \sim (\lambda x.u\,(x\,b))\,!a \rightarrow u\,(a\,b) \leftarrow (\lambda y.u\,(a\,y))\,!b \sim L$ when taking $M_1^\flat = (\lambda x.(\lambda y.u\,(x\,y))\,!b)\,!a$ in Ex. 8 as flat witness to $M_1 \sim M_4$.

Corollary 1. *$\beta!$ is confluent modulo σ.*

Proof. By the above remarks, transitivity of $\sim$ and relation algebra, ${}^*(\sim \cdot \leftarrow \cdot \sim) \cdot (\sim \cdot \rightarrow \cdot \sim)^* \subseteq \sim \cdot {}^*(\leftarrow \cdot \sim) \cdot (\sim \cdot \rightarrow)^* \cdot \sim \subseteq \sim \cdot {}^*(\multimap \cdot \sim) \cdot (\sim \cdot \multimap)^* \cdot \sim \subseteq^{\text{Thm. 2}} \sim \cdot (\sim \cdot \multimap)^* \cdot {}^*(\multimap \cdot \sim) \cdot \sim \subseteq \sim \cdot (\sim \cdot \twoheadrightarrow)^* \cdot {}^*(\twoheadleftarrow \cdot \sim) \cdot \sim \subseteq (\sim \cdot \rightarrow \cdot \sim)^* \cdot \sim \cdot {}^*(\sim \cdot \leftarrow \cdot \sim)$, entailing confluence of $[\rightarrow]$. □

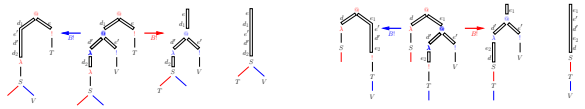


Figure 5: Joinability of *schematic* body (@–λ, left) and argument (@–!, right) $B!$-peaks.

Conclusion. We saturated $\beta!$ into $B!$ through the context-free insertion of matching @–λ-pairs in the $\beta!$-redex-pattern. Developing a set R of $B!$-redex-patterns in a term M was effectuated by a *flat development* $M \sim M^\flat \multimap N$, first transferring R into a set $R^\flat$ of $\beta!$-redex-patterns in a flat term $M^\flat \sim M$ and then developing $R^\flat$ as $M^\flat \multimap N$. They have the diamond property by $\beta!$ being orthogonal, from which confluence of bang modulo followed. We conjecture $B!$ / flat CDs are orthogonal, constitute a *residual system* [25, §8.7], suggested in Fig. 5. We have shown working with σ-equivalence classes is *effective* for the bang-calculus, but is it *efficient*?

Acknowledgments. We thank the IWC reviewers for their comments, in particular for the reference to [9, Prop. 1]. We obtained it ('Thm. 2 $\Longrightarrow$ Cor. 1') independently (in March 2024) when resolving confluence of $\beta!$ modulo σ as reported here. Though flat terms are *unblocking* in the sense of [9, Def. 1], the main problem lies elsewhere, in *proving* coherence (as flattening is *not* a function on $\sim$-classes; Ex. 8, Fig. 2). We resolved it by *saturating $\beta!$ to $B!$* and *σ-transfer*.

References

[1] B. Accattoli and D. Kesner. The structural λ-calculus. In *CSL*, LNCS, pages 381–395. Springer, 2010. doi:10.1007/978-3-642-15205-4_30.

[2] A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages*. Cambridge University Press, 1998.

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

[4] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 1984.

[5] H.P. Barendregt, R. Kennaway, J.W. Klop, and M.R. Sleep. Needed reduction and spine strategies for the lambda calculus. *Inf. Comput.*, 75(3):191–231, 1987. doi:10.1016/0890-5401(87)90001-0.

[6] N.G. de Bruijn. Generalizing Automath by means of a lambda-typed lambda calculus. In *Mathematical Logic and Theoretical Computer Science*, volume 106 of *LNPAM*, pages 71–92. CRC Press, 1986. https://pure.tue.nl/ws/portalfiles/portal/4407594/597608.pdf.

[7] A. Church and J.B. Rosser. Some properties of conversion. *TAMS*, 39:472–482, 1936. doi:10.2307/1989762.

[8] T. Ehrhard and G. Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In *PPDP*, page 174–187, 2016. doi:10.1145/2967973.2968608.

[9] T. Felicissimo. Second-order Church–Rosser modulo, without normalization. In *IWC*, 2024. URL: https://hal.science/hal-04835978/file/iwc24.pdf.

[10] G. Guerrieri and G. Manzonetto. The bang calculus and the two Girard's translations. In *TLLA*, volume 292 of *EPTCS*, pages 15–30, 2018. doi:10.4204/EPTCS.292.2.

[11] G. Guerrieri and F. Olimpieri. Categorifying non-idempotent intersection types. In *CSL*, volume 183 of *LIPIcs*, pages 25:1–25:24, 2021. doi:10.4230/LIPICS.CSL.2021.25.

[12] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, 1980. doi:10.1145/322217.322230.

[13] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM J. Comput.*, 15(4):1155–1194, 1986. doi:10.1137/0215084.

[14] D. Kapur and P. Narendran. A finite Thue system with decidable word problem and without equivalent finite canonical system. *TCS*, 35:337–344, 1985. doi:10.1016/0304-3975(85)90023-4.

[15] Z. Khasidashvili and A. Piperno. Normalization of typable terms by superdevelopments. In *CSL*, volume 1584 of *LNCS*, pages 260–282, 1999. doi:{10.1007/10703163_18}.

[16] P.B. Levy. Call-by-push-value: A subsuming paradigm. In *TLCA*, volume 1581 of *LNCS*, pages 228–243, 1999. doi:10.1007/3-540-48959-2_17.

[17] C. Marché. Normalized rewriting: an alternative to rewriting modulo a set of equations. *J. Symb. Comput.*, 21(3):253–288, 1996. doi:10.1006/jsco.1996.0011.

[18] R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *TCS*, 19(1):3–29, 1998. doi:10.1016/S0304-3975(97)00143-6.

[19] M.H.A. Newman. On theories with a combinatorial definition of "equivalence". *Ann. Math.*, 43:223–243, 1942. doi:10.2307/1968867.

[20] V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, VU Amsterdam, 1994. URL: https://research.vu.nl/files/62846778/complete%20dissertation.pdf.

[21] L. Regnier. Une équivalence sur les lambda-termes. *TCS*, 126(2):281–292, 1994. doi:10.1016/0304-3975(94)90012-4.

[22] M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[23] R.M. Smullyan. *To Mock a Mockingbird*. OUP, 2000.

[24] C.C. Squier, F. Otto, and Y. Kobayashi. A finiteness condition for rewriting systems. *TCS*, 131(2):271–294, 1994. doi:10.1016/0304-3975(94)90175-9.

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

On Completeness of the Decreasing Diagrams Method for Proving Confluence of Rewriting Systems of Cardinalities Below the First Uncountable Limit Cardinal

Ievgen Ivanov

Taras Shevchenko National University of Kyiv, Ukraine
ivanov.eugen@gmail.com

Abstract

We describe a machine-checked proof of a result in Isabelle/HOL that implies that if the cardinality of a confluent abstract rewriting system (ARS) is below the first uncountable limit cardinal, then confluence of this ARS can be proved with the help of the decreasing diagrams method using 3 labels and such an ARS has a so-called almost deterministic Church-Rosser strategy (defined by weakening conditions of the definition of deterministic one-step Church-Rosser strategies).

We also discuss consequences of this result. One consequence is that there is a statement that can be used as additional axiom to HOL that implies that every confluent ARS has an almost deterministic Church-Rosser strategy and that the decreasing diagrams method with 3 labels is complete without cardinality restrictions.

1 Introduction

An abstract rewriting system (ARS) is a pair $(A, \to)$ of a set A (a set of elements) and a binary relation $\to \subseteq A \times A$ (reduction). In the context of the usual Zermelo-Fraenkel set theory with the Axiom of Choice (ZFC) all sets are comparable in cardinality and every cardinal κ has a successor (κ^+) [3]. A (weak) limit cardinal is a cardinal that is not a successor of a cardinal. We will say that an ARS $(A, \to)$ has the cardinality κ, if the cardinality of the set A is κ.

For example, an ARS $(\mathbb{R}^2, \to')$, where $a \to' b$ if and only if a, b are real (planar) vectors such that there exists a scalar $k \in \mathbb{R}$ such that $b = k \cdot a$ has the cardinality of the continuum. If in addition to ZFC one accepts the Continuum Hypothesis (independent of ZFC, if the latter is consistent), then the ARS $(\mathbb{R}^2, \to')$ has the smallest uncountable cardinality ($\aleph_1$).

Semi-formally, an ARS $(A, \to)$ is *decreasing Church-Rosser* (*DCR*) [17, 18], if the decreasing diagrams method [17, 18] can be applied to prove that $(A, \to)$ is confluent. Here "*can be applied*" is understood as follows: there exists a labelled version of the ARS $(A, \to)$ (where labels are associated with reduction steps) with labels in some set ordered by some well-founded strict partial order $\prec$ that satisfies a condition reminiscent to local confluence, but with certain constraints on how labels of reduction steps that appear in it are related by $\prec$.

A rigorous definition of the class of DCR ARS can be found in [18].

In [18] V. van Oostrom showed that every DCR ARS is confluent (the decreasing diagrams method is sound for proving confluence) and also showed that every ARS with the cofinality property is DCR (the method is complete for proving confluence of such ARS). Countable confluent ARS have the cofinality property [8], but uncountable ARS may *not* have it.

In [1] J. Endrullis, J.W. Klop, R. Overbeek considered restricted versions of related questions and proposed a hierarchy of classes DCR_α, where for each ordinal α, DCR_α is the class of all ARS that can be proved confluent using the decreasing diagrams method under the assumption that the set of labels and the order $\prec$ on labels are fixed as follows:

- the set of labels is the set of all ordinals below α
- the order $\prec$ is the restriction of the usual strict order on ordinals to the set of labels.

In [1] it was shown that the class DCR_2 contains all countable confluent ARS. In essence, this result is closely related to the existence of deterministic one-step Church-Rosser strategies [6, Definition 3] in countable confluent ARS. However, such strategies may not exist in uncountable confluent ARS [6, Proposition 2]. In [5] it was shown that the class DCR_3 contains all confluent ARS of the smallest uncountable cardinality ($\aleph_1$), but the same does *not* hold for DCR_2. This result can be considered as a consequence of a stronger result obtained in [6, Theorem 1] that shows an ARS of the cardinality $\aleph_1$ is confluent if and only if it has a so-called 1-*branching one-step nondeterministic* Church-Rosser strategy [6]. Informally, the latter is a weakening of the concept of a deterministic one-step Church-Rosser strategy that admits a minimum amount of nondeterminism of a strategy. In this paper we will alternatively call such strategies *almost deterministic Church-Rosser strategies*.

In this paper we extend the above mentioned results and show that every confluent ARS of the cardinality $\aleph_k$ for finite k ($\aleph_2$, $\aleph_3$, etc.) has an almost deterministic Church-Rosser strategy. This result implies that

1. the class DCR_3 contains confluent ARS of the cardinalities $\aleph_k$ for all finite k
2. there is a sentence that we will call *A-Card* below that can be used as additional axiom to HOL such that HOL + *A-Card* proves that every confluent ARS has an almost deterministic Church-Rosser strategy and ZF proves consistency of HOL + *A-Card*.

2 Preliminaries

The reader can assume that a background theory for the natural language propositions given below is Zermelo-Fraenkel set theory with the Axiom of Choice (ZFC) or Von Neumann-Bernays-Gödel (NBG) set theory [13, 3] with the axiom of global choice [3, p. 133] (that is useful for reasoning about proper classes). Formal statements and proofs from Isabelle/HOL formalization [7] that contains a supplementary material for this paper can be transferred to the set-theoretic setting using a process similar to the one described in [10, 12, 15].

A variant of explicit definition of the classes DCR_α based on [2, Definition 4.4] and [2, Definition 4.2] is given below. For any ordinal α denote as $\Upsilon\alpha$ the set of all ordinals less than α (under the usual set-theoretic definition of ordinals this notation is trivial, i.e. $\Upsilon\alpha$ is the ordinal α itself, but it will be used to highlight that elements of α have the role of labels).

Definition 2.1 ([4]). *Let γ be an ordinal. An ARS $(A, \to)$ belongs to the class DCR_γ, if there exists an indexed family $(\to_\alpha)_{\alpha\in(\Upsilon\gamma)}$ of binary relations on A indexed by ordinals $\alpha < \gamma$ such that $\to = \bigcup_{\alpha<\gamma} \to_\alpha$ and for every ordinals $\alpha, \beta \in (\Upsilon\gamma)$ and for every $a, b, c \in A$ the following holds: if $a \to_\alpha b \wedge a \to_\beta c$, then there exist elements $b', b'', c', c'', d \in A$ such that*

$$\left(b \underset{\Upsilon\alpha}{\overset{*}{\dashrightarrow}} b' \underset{\{\beta\}}{\overset{=}{\dashrightarrow}} b'' \underset{\Upsilon\alpha\cup\Upsilon\beta}{\overset{*}{\dashrightarrow}} d\right) \wedge \left(c \underset{\Upsilon\beta}{\overset{*}{\dashrightarrow}} c' \underset{\{\alpha\}}{\overset{=}{\dashrightarrow}} c'' \underset{\Upsilon\beta\cup\Upsilon\alpha}{\overset{*}{\dashrightarrow}} d\right),$$

where symbols of the form $\underset{K}{\overset{=}{\dashrightarrow}}$ and $\underset{K}{\overset{*}{\dashrightarrow}}$, where K is an expression, mean:

$$\underset{K}{\overset{=}{\dashrightarrow}} = \{(a,a) \mid a \in A\} \cup \bigcup_{\kappa\in K} \to_\kappa \quad \text{and} \quad \underset{K}{\overset{*}{\dashrightarrow}} = \{(a,a) \mid a \in A\} \cup \left(\bigcup_{\kappa\in K} \to_\kappa\right)^+,$$

and the symbol $(\cdot)^+$ *is used to denote the transitive closure of a binary relation.*

Note that:

- for any ordinal α, every ARS in DCR_α is decreasing Church-Rosser in the sense of [18]
- $DCR_\alpha \subseteq DCR_\beta$ whenever $\alpha \leq \beta$.

An explicit definition of the class DCR_3 can be found in [4, Proposition 2].
In this paper we will also use the following notations and definitions used in [6].

Definition 2.2 ([16, 6])**.** *A one-step (reduction) strategy for an ARS* $(X, \to)$ *is a subrelation* $\to_s \subseteq \to$ *such that* $NF(X, \to_s) = NF(X, \to)$*, or, equivalently, if* $dom(\to) = dom(\to_s)$*.*

Definition 2.3 (based on [9, Definition 6])**.** *A deterministic one-step Church-Rosser strategy for an ARS* $(X, \to)$ *is a one-step strategy* $\to_s$ *for* $(X, \to)$ *such that* $\to_s$ *is functional and*

$$\forall a, b \in X \ (a \leftrightarrow^* b \ \Rightarrow \ \exists c \in X \ a \to_s^* c \wedge b \to_s^* c).$$

For every one-step strategy $\to_s$ for the ARS $(X, \to)$ the *branching weight function* $bw_{\to_s}$ (parameterized by $\to_s$) is defined as follows [6]:

$$bw_{\to_s}((a,b)) = \begin{cases} 0, & \text{if } \{b' \mid a \to_s b'\} = \{b\}, \\ 1, & \text{if } \{b' \mid a \to_s b'\} \neq \{b\}, \end{cases}$$

i.e. $bw_{\to_s}$ maps a rewrite step $(a, b) \in \to_s$ to 0, if b is a unique direct successor of a in $\to_s$, and maps (a, b) to 1 otherwise.

For a binary relation r denote as $dom(r)$ its domain, i.e. $\{a \mid \exists b \ (a, b) \in r\}$.

For an ARS $(X, \to)$ denote $NF(X, \to) = X \backslash dom(\to)$ (the set of normal forms).

Definition 2.4 ([6])**.** *For* $k \in \mathbb{N}_0$*, a* k*-branching one-step nondeterministic Church-Rosser (or* k*-branching 1-nCR) strategy for an ARS* $(X, \to)$ *is a one-step strategy* $\to_s$ *for* $(X, \to)$ *such that*

$$\forall a, b \in X \ (a \leftrightarrow^* b \Leftrightarrow B_k(a) \cap B_k(b) \neq \emptyset \ ),$$

where

$$B_k(x) = \{ \ y \in X \mid \exists n \in \mathbb{N}_0 \ \exists a_0, a_1, ..., a_n \in X \quad x = a_0 \to_s a_1 \to_s ... \to_s a_n = y \ \wedge \\ \wedge \ |\{i \in \{0, 1, ..., n-1\} \mid \exists b, b' \ (b \neq b' \wedge a_i \to_s b \wedge a_i \to_s b')\}| \leq k \ \}$$

and $|.|$ *is the cardinality of a set, i.e.* $B_k(x)$ *is the ball of radius* k *in* $(X, \to_s)$ *at an element* x *w.r.t. the weight function* $bw_{\to_s}$*.*

Definition 2.5. *An almost deterministic Church-Rosser (CR) strategy for an ARS* $(X, \to)$ *is a* 1*-branching 1-nCR strategy for* $(X, \to)$*.*

3 Main Results

For any set S the *equinumerosity* relation $\sim_S$ on the power set of S is defined as follows:

$$X \sim_S Y \text{ if and only if } X, Y \subseteq S \text{ and there exists a bijection } f : X \to Y.$$

Theorem 1. *Let r be a binary relation on a non-empty set U.*

Assume that the set of equivalence classes of infinite subrelations of r w.r.t. $\sim_r$ is finite.

Then (U, r) is confluent if and only if (U, r) has an almost deterministic (i.e. 1-branching 1-step nondeterministic) Church-Rosser strategy.

The statement and machine-checked proof of a formalized version of Theorem 1 in the formal language of Isabelle/HOL [14] is given in the supplementary material [7, lines 13095-13101].

Theorem 2. *Let r be a binary relation on a non-empty set U. Assume that*

1. *the set of equivalence classes of infinite subrelations of r w.r.t. $\sim_r$ is finite*
2. *r is confluent.*

Then the ARS (U, r) belongs to the class DCR_3.

The statement and machine-checked proof of a formalized version of Theorem 2 in the formal language of Isabelle/HOL [14] is given in the supplementary material [7, lines 13109-13150].

The statement has the following form (where ... denotes a continuation not shown here):

```
theorem thm_main:
fixes r::"('U×'U) set"
assumes a_card: "finite { E. ∃ X ⊆ r. ¬ finite X ∧
                                        E = {Y. Y ⊆ r ∧ (∃ f. bij_betw f X Y) } }"
assumes a_confl: "∀ a b c. (a,b) ∈ r^* ∧ (a,c) ∈ r^* ⟶
                     (∃ d. (b,d) ∈ r^* ∧ (c,d) ∈ r^*)"
shows "∃ r0 r1 r2 . (
         ( r = (r0 ∪ r1 ∪ r2) )
       ∧ ( ∀ a b c. (a,b) ∈ r0 ∧ (a,c) ∈ r0 ⟶ (∃ d. (b,d) ∈ r0^= ∧ (c,d) ∈ r0^= ) )
              ...
```

The formal assumptions `a_card` and `a_confl` represent the conditions 1, 2 of Theorem 2 respectively. The conclusion (**shows**) represents the conditions of Definition 2.1 for the ARS (U, r) and $\gamma = 3$, where the relations $\to$, $\to_0$, $\to_1$, $\to_2$ in the notation of Definition 2.1 are denoted as `r`, `r0`, `r1`, `r2` in Isabelle/HOL formalization.

The high-level structure of the proof of Theorem 2 given in [7] is similar to the structure of the proof of [5, Theorem 18] that establishes that DCR_3 contains all confluent ARS with reduction relation of the cardinality $\aleph_1$. In essence, the proof proceeds by constructing an almost deterministic Church-Rosser strategy $\to_s$ for a confluent ARS of cardinality $\aleph_k$ (for finite k) and labeling deterministic and nondeterministic steps in a strategy with 0, 1 respectively, and labeling steps outside a strategy with 2. In more detail, the proof proceeds through the steps described in [5, Subsection 5.1], however the implementation of the sub-step B1b in [7] (supplementary material for this paper) differs from the implementation of this sub-step in [5], because Lemma 27 from [5] cannot be applied when the relations $r|_{W_\alpha}$ from its assumption lack the cofinality property. In [7] this lemma is replaced with a less restrictive formal lemma `lem_cfcomp_d2uset_cs` [7, lines 11294-11310] that does not require its user to pre-define pseudo-trees $(R_\alpha)_{\alpha \in S}$, and instead constructs the required trees "dynamically" in the proof. In this way the assumptions of `lem_cfcomp_d2uset_cs` no longer imply the cofinality property for $r|_{W_\alpha}$ and the lemma can be applied in all situations necessary for the proof of Theorem 2. Technical details of the proof can be found in [7, lines 11294-11784].

Corollary 1. *Let $(A, \to)$ be a confluent ARS that has cardinality $\aleph_k$ for some non-negative integer k. Then $(A, \to)$ has an almost deterministic Church-Rosser strategy and $(A, \to)$ belongs to the class DCR_3.*

Proof. Note that $|\rightarrow| \leq |A \times A| \leq \aleph_k$. Then infinite subrelations of $\rightarrow$ have no more than $k+1$ different cardinalities, so the assumption of Theorem 1 and conditions 1, 2 of Theorem 2 are satisfied. Then $(A, \rightarrow)$ has an almost deterministic Church-Rosser strategy and $(A, \rightarrow)$ belongs to the class DCR_3 by Theorem 1 and 2. □

4 Discussion

4.1 Dependence of Formalization on Isabelle/HOL Library

The statements of the formalized versions of Theorems 1 and 2 in [7] use several notions formalized in Isabelle/HOL's library such as relations (`rel`), finiteness (`finite`), bijection (`bij_betwj`), reflexive closure $r\hat{}^{=}$, reflexive transitive closure $r\hat{}^{*}$.

However, one can restate these theorems without using such library notions, using the syntax given in the standard HOL description [15] by A. Pitts as follows:

1. characterize reflexive transitive closure of a relation as the least reflexive transitive relation that includes the given one
2. replace set finiteness condition with a variant of formulation of Dedekind-finiteness condition (equivalent to finiteness in Isabelle/HOL)
3. represent sets using indicator functions.

For example, elimination of Isabelle/HOL's library reflexive transitive closure operation from the statement of `thm_main` from [7] can be based on the following lemma (that can be straightforwardly proved in Isabelle/HOL):

```
definition "preord r ≡ ((∀ c. (c,c) ∈ r) ∧
                        (∀ a b c. (a,b) ∈ r ∧ (b,c) ∈ r ⟶ (a,c) ∈ r ))"

lemma
fixes P::"'U rel ⇒ bool" and r::"'U rel"
shows "P (r^*) = ( ∀ t. ( r ⊆ t ∧ preord t ∧
                  (∀ r'. r ⊆ r' ∧ preord r' ⟶ t ⊆ r') ⟶ P t ) )"
```

Counterparts of set inclusion, image under function, equinumerosity, and Dedekind-finiteness for indicator functions can be defined, respectively, as follows (where name prefix I is used to highlight indicator functions):

```
definition incl where "incl IA IB ≡ (∀ x. IA x ⟶ IB x)"
definition img where "img f IA ≡ (λ y. ∃ x. IA x ∧ y = f x)"
definition equinum where "equinum IA IB ≡
                              (∃ f g. incl IA (img f IB) ∧ incl IB (img g IA))"
definition fin where "fin IA ≡ (∀ I. incl I IA ∧ equinum I IA ⟶ I = IA)"
```

In particular, using definitions of `incl`, `img`, `equinum`, `fin` given above, it is straightforward to prove in Isabelle/HOL the following equivalence between the universal closure of the cardinality assumption (`a_card`) of `thm_main` from [7] and the universal closure of its reformulation that does not reference Isabelle-specific library symbols:

```
lemma "(∀ r::('U rel). finite { E. ∃ X ⊆ r. ¬ finite X
                    ∧ E = {Y. Y ⊆ r ∧ (∃ f. bij_betw f X Y) } })
      =
```

```
(∀ Ir::((bool ⇒ 'U) ⇒ bool).
     fin (img (λI I'. incl I' Ir ∧ equinum I I')
         (λI. incl I Ir ∧ ¬ fin I)))"
```

In [11] O. Kunčar and A. Popescu showed proof-theoretic conservativity of Isabelle/HOL with constant and type definitions (for recent versions of Isabelle) over initial HOL. This implies that after elimination of Isabelle/HOL-specific library symbols from the statements of the formalized versions of Theorems 1 and 2 in [7] as described above, Isabelle/HOL-specific library symbols can also be eliminated from the proofs of these theorems.

4.2 Consistency of Unconditional Existence of Almost Deterministic Church-Rosser Strategies and Completeness of Decreasing Diagrams Method

In the set-theoretic semantics of HOL described by A. Pitts in [15], types are interpreted as n-ary operations on a fixed set $\mathcal{U}$ (universe of sets) that satisfies certain conditions. In particular [15, Section 1.1], $\mathcal{U}$ can be taken to be $V_{\omega+\omega}\backslash\{\emptyset\}$, where $V_{\omega+\omega}$ is the respective stage in von Neumann cumulative hierarchy in the sense of ZFC set theory. In the latter case, under the Generalized Continuum Hypothesis (GCH) assumption the interpretation of the `a_card` assumption mentioned in Section 4.1 holds true regardless of a set in $\mathcal{U}$ assigned to `'U`. Thus an equivalent reformulation of the universal closure of `a_card` that does not depend on Isabelle/HOL-specific library (as described in Subsection 4.1) can be taken as an additional axiom to HOL. Will call this axiom *A-Card*.

Taking into account soundness theorem for HOL [15, Section 2.3.2], one can conclude that ZFC+GCH proves that HOL with *A-Card* has a (standard) model and is consistent, i.e. F (False) cannot be derived from it [15, Section 2.4.5]. Note that since consistency is defined syntactically, from Shoenfield's absoluteness theorem it follows that ZF proves consistency of HOL with *A-Card* as well. Also, from Theorems 1 and 2 it follows that in HOL with *A-Card* one can derive that every confluent ARS has an almost deterministic Church-Rosser strategy and that the decreasing diagrams method is complete for proving confluence without cardinality restrictions (assuming that these statements are appropriately formulated in the language of HOL, e.g as described in Section 4.1).

5 Conclusion and Future Work

We have discussed consequences of the main results.

Future work includes investigation of generalizations of Church-Rosser strategies for ARS of high uncountable cardinality and investigation of practical implications of the obtained results.

References

[1] Jörg Endrullis, Jan Willem Klop, and Roy Overbeek. Decreasing diagrams with two labels are complete for confluence of countable systems. In *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, pages 14:1–14:15, 2018.

[2] Jörg Endrullis, Jan Willem Klop, and Roy Overbeek. Decreasing diagrams for confluence and commutation. *Logical Methods in Computer Science*, 16:23:1–23:25, 2020.

[3] A.A. Fraenkel, Y. Bar-Hillel, and A. Levy. *Foundations of Set Theory*. Elsevier, 1973.

[4] Ievgen Ivanov. On non-triviality of the hierarchy of decreasing Church-Rosser abstract rewriting systems. In *Proceedings of the 13th International Workshop on Confluence. July 9, 2024, Tallinn, Estonia*, pages 30–35.

[5] Ievgen Ivanov. Completeness of the decreasing diagrams method for proving confluence of rewriting systems of the least uncountable cardinality. *Leibniz international proceedings in informatics*, 337:25:1–25:20, 2025. https://doi.org/10.4230/LIPIcs.FSCD.2025.25.

[6] Ievgen Ivanov. Generalization of Church-Rosser strategies for confluent abstract rewriting systems of the smallest uncountable cardinality. In *16th International Workshop on Rewriting Logic and its Applications (WRLA 2026)*, 2026.

[7] Ievgen Ivanov. Supplementary material for this paper (formal theory DCRNk for Isabelle/HOL). https://doi.org/10.5281/zenodo.20057503, 2026.

[8] Jan Willem Klop. *Combinatory Reduction Systems*. PhD thesis, Rijks-universiteit Utrecht, 1980.

[9] J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Reduction strategies and acyclicity. volume 4600 of *Lecture Notes in Computer Science*, pages 89–112, 2007.

[10] Alexander Krauss and Andreas Schropp. A mechanized translation from Higher-Order Logic to set theory. volume 6172 of *Lecture Notes in Computer Science*, pages 323–338. Springer, 2010.

[11] Ondřej Kunčar and Andrei Popescu. Safety and conservativity of definitions in HOL and Isabelle/HOL. *Proceedings of the ACM on Programming Languages*, 2:1–26, 2018.

[12] Ondřej Kunčar and Andrei Popescu. A consistent foundation for Isabelle/HOL. *Journal of Automated Reasoning*, 62:531–555, 2019.

[13] Elliott Mendelson. *Introduction to Mathematical Logic*. CRC Press, 2015.

[14] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

[15] Andrew Pitts. The HOL logic. In *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*, pages 191–232. Cambridge University Press, 1993.

[16] Y. Toyama. Reduction strategies for left-linear term rewriting systems. volume 3838 of *Lecture Notes in Computer Science*, pages 198–223, 2005.

[17] Vincent Van Oostrom. Confluence by decreasing diagrams. *Theoretical computer science*, 126(2):259–280, 1994.

[18] Vincent Van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit Amsterdam, 1994.

Disproving Reachability in Probabilistic Term Rewriting

Jan-Christoph Kassing[1][2], Moritz Leven Rosarius[1][3],
Henri Nagel[1][4], and Jürgen Giesl[1][5]

[1] RWTH Aachen University, Aachen, Germany
[2] kassing@cs.rwth-aachen.de
[3] leven@cs.rwth-aachen.de
[4] henri.nagel@rwth-aachen.de
[5] giesl@informatik.rwth-aachen.de

Abstract

Reachability is a central question in term rewriting: can a given target term (e.g., an error state) be reached from a start term? It is also an important property in confluence analysis, and corresponding tools compete in the annual confluence competition. An interesting generalization of this problem is handling programs that can make random choices during execution. For such probabilistic programs, reachability becomes a *quantitative* property instead of a *qualitative* one: instead of asking *whether* the target is reachable, one asks *with which probability* it is reached. We lift reachability analysis from ordinary term rewriting to probabilistic term rewrite systems. To do so, we formalize the maximal probability of reaching a target term and adapt two techniques for analyzing reachability (based on symbol transition graphs and on term orderings) to compute upper bounds on this probability.

1 Introduction

Term rewrite systems (TRSs) are a fundamental model for program execution, transformation, and verification. A TRS consists of a finite number of rewrite rules $\ell \to r$, indicating that parts of a term that are matched by ℓ can be replaced by the reduct r. A central question is the *reachability problem* "$s \to_{\mathcal{R}}^* t$?": can a target t (e.g., an error state) be reached from a start term s? Often, s and t are templates, and one asks whether t can be reached for *some* ground instantiation σ of their variables, i.e., "$\exists \sigma.\, s\sigma \to_{\mathcal{R}}^* t\sigma$?". Reachability for TRSs has been studied extensively, e.g., [6, 8, 9, 10].

We lift reachability analysis to probabilistic term rewrite systems (PTRSs), where the rule selection remains non-deterministic, but after selecting a rule, the reduct is chosen probabilistically. We formalize the maximal probability of reaching a target (Sect. 2) and adapt two existing techniques from the non-probabilistic setting to compute upper bounds on it. This yields upper bounds on the maximal reachability probability: we can prove that a target is reached with probability at most p, and thereby disprove reachability claims (e.g., almost-sure reachability whenever $p < 1$, or reachability at all whenever $p = 0$). First, the symbol transition graph turns into an over-approximating Markov decision process (MDP), whose reachability probability can be computed with tools like Storm [7] (Sect. 3). Second, we lift term orderings for disproving reachability to the probabilistic setting (Sect. 4). We plan to implement both techniques in AProVE [5] and to evaluate them experimentally.

2 Preliminaries

We assume the reader to be familiar with ordinary term rewriting as presented, e.g., in [1]. In the probabilistic setting, a single reduction step leads to a *finite multi-distribution* over possible

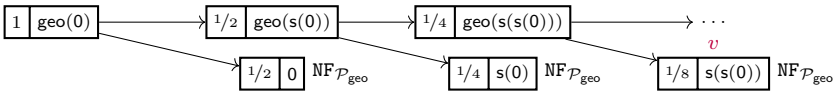


Figure 1: A $\mathcal{P}_{\mathsf{geo}}$-RT $\mathfrak{T}$ with $\mathrm{root}(\mathfrak{T}) = \mathsf{geo}(\mathsf{0})$. The node v is labeled by the probability $p_v = 1/8$ and the term $a_v = \mathsf{s}(\mathsf{s}(\mathsf{0}))$.

results rather than to a single result. A finite *multi-distribution* μ on a set $A \neq \varnothing$ is a finite multiset of pairs ${}^{p}a$, where $0 < p \leq 1$ is a probability and $a \in A$, such that $\sum_{{}^{p}a \in \mu} p = 1$. Let $\mathrm{FDist}(A)$ denote the set of all finite multi-distributions on A. For $\mu \in \mathrm{FDist}(A)$, its *support* is the multiset $\mathrm{Supp}(\mu) = \{a \mid {}^{p}a \in \mu\}$. A *probabilistic abstract reduction system* (PARS) is a pair $(A, \to)$ such that $\to \subseteq A \times \mathrm{FDist}(A)$.

Let $\mathcal{T} = \mathcal{T}(\Sigma, \mathcal{V})$ be the set of terms over a finite signature Σ and variable set $\mathcal{V}$. A *probabilistic term rewrite rule* $\ell \to \mu$ is a pair $(\ell, \mu) \in \mathcal{T} \times \mathrm{FDist}(\mathcal{T})$ such that $\ell \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ for all $r \in \mathrm{Supp}(\mu)$, and a *probabilistic TRS* (PTRS) is a finite set of probabilistic term rewrite rules. Similar to TRSs, a PTRS $\mathcal{P}$ induces a PARS $(\mathcal{T}, \to_{\mathcal{P}})$ with $s \to_{\mathcal{P}} \{{}^{p_1}t_1, \ldots, {}^{p_k}t_k\}$ if there is a position $\pi \in \mathrm{Pos}(s)$, a rule $\ell \to \{{}^{p_1}r_1, \ldots, {}^{p_k}r_k\} \in \mathcal{P}$, and a substitution σ such that $s|_\pi = \ell\sigma$ and $t_j = s[r_j\sigma]_\pi$ for all $1 \leq j \leq k$. Often, we simply refer to $\mathcal{P}$ instead of $\to_{\mathcal{P}}$. Consider the PTRS $\mathcal{P}_{\mathsf{geo}}$ with the only rule $\mathsf{geo}(x) \to \{{}^{1/2}\mathsf{geo}(\mathsf{s}(x)),\ {}^{1/2}x\}$. When starting with $\mathsf{geo}(\mathsf{0})$, repeated rewriting yields $\mathsf{s}^k(\mathsf{0})$ with a probability of $(1/2)^{k+1}$, i.e., a geometric distribution.

To track rewrite sequences of a PARS $(A, \to)$ with their probabilities, we consider *reduction trees (RTs)*. The nodes v of a $\to$-RT are labeled by pairs ${}^{p_v}a_v$ of a probability $p_v \in (0, 1]$ and an object $a_v \in A$, where the probability at the root is 1. For each node v with successors $w_1, \ldots, w_k$, the edge relation represents a rewrite step, i.e., $a_v \to \{{}^{p_{w_1}/p_v}a_{w_1}, \ldots, {}^{p_{w_k}/p_v}a_{w_k}\}$. For a $\to$-RT $\mathfrak{T}$, $V(\mathfrak{T})$ denotes its set of nodes, $\mathrm{root}(\mathfrak{T})$ is the object at its root, and $\mathrm{Leaf}(\mathfrak{T})$ denotes its set of leaves. An example for a $\mathcal{P}_{\mathsf{geo}}$-RT is shown in Fig. 1.

To analyze the probability of a term s reaching a term t, we use *truncated* RTs. For an RT $\mathfrak{T}$ and a term u, the truncated RT $\mathfrak{T}_{|u}$ results from $\mathfrak{T}$ by removing all proper successors of every node v labeled with a pair ${}^{p_v}u$. Hence, every node labeled with u is a leaf of $\mathfrak{T}_{|u}$, and along each path of $\mathfrak{T}_{|u}$ there is at most one node labeled with u, which is then the last node of that path. Intuitively, this lets us collect the probability mass that reaches u *for the first time*, without counting paths that pass through u several times.

Definition 1 (Reachability Probability)**.** Let $\mathcal{P}$ be a PTRS and $s, t \in \mathcal{T}$. For a $\mathcal{P}$-RT $\mathfrak{T}$, the *probability of reaching t from s in $\mathfrak{T}$* is $\mathbb{P}_{\mathfrak{T}}(s \to_{\mathcal{P}}^{*} t) = \sum_{\substack{v \in \mathrm{Leaf}(\mathfrak{T}_{|t}) \\ a_v = t}} p_v$ if $\mathrm{root}(\mathfrak{T}) = s$ and $\mathbb{P}_{\mathfrak{T}}(s \to_{\mathcal{P}}^{*} t) = 0$ otherwise. The *maximal reachability probability* of reaching t from s is $\mathbb{P}^{\max}(s \to_{\mathcal{P}}^{*} t) = \sup_{\mathcal{P}\text{-RT } \mathfrak{T}} \mathbb{P}_{\mathfrak{T}}(s \to_{\mathcal{P}}^{*} t)$. For a PTRS $\mathcal{P}$ and terms $s, t \in \mathcal{T}$, we write $s \dashrightarrow_{\leq p}^{\mathcal{P}} t$ iff $\mathbb{P}^{\max}(s\sigma \to_{\mathcal{P}}^{*} t\sigma) \leq p$ holds for *every* ground substitution σ w.r.t. s and t.

The supremum in $\mathbb{P}^{\max}(s \to_{\mathcal{P}}^{*} t)$ ranges over all $\mathcal{P}$-RTs with root ${}^{1}s$, i.e., over all ways of resolving the nondeterministic choices. Thus, $\mathbb{P}^{\max}(s \to_{\mathcal{P}}^{*} t)$ is the tightest upper bound on the probability of reaching t from s under any evaluation strategy, and $s \dashrightarrow_{\leq p}^{\mathcal{P}} t$ states that this bound stays below p for *all* ground instantiations of the variables of s and t. As an example, reconsider $\mathcal{P}_{\mathsf{geo}}$ and the RT $\mathfrak{T}$ in Fig. 1. Its leaves are the normal forms $\mathsf{s}^k(\mathsf{0})$, each reached with probability $(1/2)^{k+1}$. Hence $\mathbb{P}^{\max}(\mathsf{geo}(\mathsf{0}) \to_{\mathcal{P}_{\mathsf{geo}}}^{*} \mathsf{s}(\mathsf{0})) = 1/4$. The term $\mathsf{s}(\mathsf{0})$ only occurs once as a leaf (with probability $1/4$), and it cannot be reached again once we have rewritten to $\mathsf{geo}(\mathsf{s}(\mathsf{s}(\mathsf{0})))$ or stopped at $\mathsf{0}$. Moreover, there is only one $\mathcal{P}_{\mathsf{geo}}$-RT with root ${}^{1}\mathsf{geo}(\mathsf{0})$. We therefore have $\mathsf{geo}(\mathsf{0}) \dashrightarrow_{\leq 1/4}^{\mathcal{P}_{\mathsf{geo}}} \mathsf{s}(\mathsf{0})$, and this bound is tight. Similarly, $\mathsf{geo}(x) \dashrightarrow_{\leq 1/2}^{\mathcal{P}_{\mathsf{geo}}} x$ holds.

2

3 Upper Bounds on Reachability via MDPs

We now lift the *symbol transition graph* technique of [9] to the probabilistic setting. The idea is to over-approximate the (in general infinite) rewrite relation $\to_{\mathcal{P}}$ by a *finite* MDP and to read off an upper bound on $\sup_\sigma \mathbb{P}^{\max}(s\sigma \to_{\mathcal{P}}^* t\sigma)$ from the maximal reachability probability of that MDP, computable by probabilistic model checkers such as Storm [7]. The abstraction has two ingredients: (i) we only keep the top of each term, cutting everything below a fixed height n, so that finitely many terms remain; and (ii) since a cut term stands for *all* of its instances, a rule may apply to some instance even if its left-hand side does not match the cut term, so we replace matching by *unification*, i.e., the edges of the MDP are *probabilistic narrowing steps*. The *height* of a term is $h(x) = 0$ for $x \in \mathcal{V}$ and $h(f(t_1, \dots, t_k)) = 1 + \max\{h(t_1), \dots, h(t_k)\}$.

Definition 2 (Cut)**.** For $t \in \mathcal{T}$ and $n \in \mathbb{N}$, the *cut* $\lceil t \rceil_n$ replaces every subterm $t|_\pi$ at a position $\pi \in \mathrm{Pos}(t)$ with $|\pi| = n$ by a fresh variable (distinct positions get distinct fresh variables).

Thus $h(\lceil t \rceil_n) \le n$, and $\lceil t \rceil_n = t$ (up to renaming) if and only if $h(t) \le n$. Up to renaming there are only finitely many terms of height at most n, and a cut term u represents the set $\gamma(u) = \{u\sigma \mid \sigma \text{ a substitution}\}$ of all its instances (so $t \in \gamma(\lceil t \rceil_n)$ for all terms t), e.g., $\lceil \mathsf{geo}(\mathsf{s}(\mathsf{s}(\mathsf{0})))\rceil_2 = \mathsf{geo}(\mathsf{s}(x))$.

Recall that an MDP $\mathcal{M} = (S, Act, \mathbb{P})$ consists of a countable set of *states* S, finitely many *actions* Act, and a *transition probability function* $\mathbb{P}\colon S \times Act \times S \to [0,1]$ with $\sum_{s' \in S} \mathbb{P}(s, \alpha, s') \in \{0, 1\}$ for all states $s \in S$ and actions $\alpha \in Act$. The action α is *enabled* in s if this sum is 1. A *scheduler* chooses an enabled action in each state (possibly depending on the history), inducing a probability measure on the runs of $\mathcal{M}$. For a set $T \subseteq S$ of *target* states, $\mathbb{P}^{\max}_{\mathcal{M}}(s, T)$ denotes the maximal probability of eventually reaching T from s over all schedulers.

Definition 3 (n-Cut MDP)**.** Let $\mathcal{P}$ be a PTRS with maximal arity m and $n \in \mathbb{N}$. The *n-cut MDP* $\mathcal{M}_n(\mathcal{P}) = (S_n, Act, \mathbb{P})$ has states $S_n = \{t \in \mathcal{T} \mid h(t) \le n\}$ (up to renaming) and actions $Act = \{1, \dots, m\}^{\le n} \times \mathcal{P}$ (a position together with a rule, representing a possible rewrite step). An action $(\pi, \ell \to \mu)$ with $\mu = \{{}^{p_1}r_1, \dots, {}^{p_k}r_k\}$ is *enabled* in a term t (where w.l.o.g. $\mathcal{V}(t) \cap \mathcal{V}(\ell) = \varnothing$) iff $\pi \in \mathrm{Pos}(t)$ and $t|_\pi$ unifies with ℓ with mgu σ. It then performs a probabilistic narrowing step followed by a cut, yielding $\mathbb{P}(t, (\pi, \ell \to \mu), s') = \sum_{1 \le j \le k,\, \lceil (t\sigma)[r_j\sigma]_\pi \rceil_n = s'} p_j$ and $\mathbb{P}(t, (\pi, \ell \to \mu), \cdot) = 0$ if it is not enabled.

As an example, the 2-Cut MDP $\mathcal{M}_2(\mathcal{P}_{\mathsf{geo}})$ is shown in Fig. 2. Here, we only present the nodes that are reachable from $\mathsf{geo}(\mathsf{0})$.

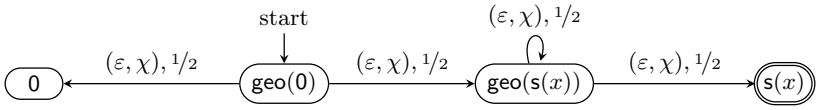


Figure 2: The reachable part of $\mathcal{M}_2(\mathcal{P}_{\mathsf{geo}})$ for $\mathcal{P}_{\mathsf{geo}}$ (both actions apply the rule at the root). A double border marks the target $\mathsf{s}(x) \in T^2_{\mathsf{s}(\mathsf{0})}$, and $\mathsf{0}$ is a normal form. By χ we denote the only rule in $\mathcal{P}_{\mathsf{geo}}$, namely $\mathsf{geo}(x) \to \{{}^{1/2}\mathsf{geo}(\mathsf{s}(x)),\ {}^{1/2}x\}$.

Applying the mgu σ to the *whole* term t (not only to the redex $t|_\pi$) is precisely a narrowing step $t \rightsquigarrow (t\sigma)[r_j\sigma]_\pi$, since σ may also instantiate variables of t outside of π. To obtain a bound on $\sup_\sigma \mathbb{P}^{\max}(s\sigma \to_{\mathcal{P}}^* t\sigma)$, we start in $\lceil s \rceil_n$ and use the target set $T^n_t = \{t' \in S_n \mid t' \text{ and } \lceil t \rceil_n \text{ are unifiable}\}$, i.e., all cut terms with a common instance with t.

Theorem 4 (Upper Bounds on $\sup_\sigma \mathbb{P}^{\max}(s\sigma \to_{\mathcal{P}}^* t\sigma)$)**.** *Let $\mathcal{P}$ be a PTRS, $s, t \in \mathcal{T}$, and $n \in \mathbb{N}$. Then*

$$\mathbb{P}^{\max}_{\mathcal{M}_n(\mathcal{P})}(\lceil s \rceil_n, T_t^n) \leq p \quad \Longrightarrow \quad \sup_\sigma \mathbb{P}^{\max}(s\sigma \to_{\mathcal{P}}^* t\sigma) \leq p$$

Proof Sketch. Fix a substitution σ and a $\to_{\mathcal{P}}$-RT $\mathfrak{T}$ with $\mathrm{root}(\mathfrak{T}) = s\sigma$, and map every node labeled with a term u to the state $\lceil u \rceil_n$. A concrete step at a position π with $|\pi| < n$ corresponds to the action $(\pi, \ell \to \mu)$ with the *same* probabilities p_j, while a step with $|\pi| \geq n$ occurs inside a cut-off subterm and leaves $\lceil u \rceil_n$ unchanged. So $\mathfrak{T}$ induces a scheduler of $\mathcal{M}_n(\mathcal{P})$ starting in $\lceil s\sigma \rceil_n$ that reaches T_t^n with probability at least $\mathbb{P}_{\mathfrak{T}}(s\sigma \to^* t\sigma)$, as every leaf labeled $t\sigma$ maps to $\lceil t\sigma \rceil_n \in T_t^n$; thus $\mathbb{P}_{\mathfrak{T}}(s\sigma \to^* t\sigma) \leq \mathbb{P}^{\max}_{\mathcal{M}_n(\mathcal{P})}(\lceil s\sigma \rceil_n, T_t^n)$. Since $\lceil s\sigma \rceil_n \in \gamma(\lceil s \rceil_n)$ and a more general state over-approximates its instances (every narrowing step enabled in an instance is also enabled, up to instantiation, in the more general term), we have $\mathbb{P}^{\max}_{\mathcal{M}_n(\mathcal{P})}(\lceil s\sigma \rceil_n, T_t^n) \leq \mathbb{P}^{\max}_{\mathcal{M}_n(\mathcal{P})}(\lceil s \rceil_n, T_t^n) \leq p$. Taking the supremum over all $\mathfrak{T}$ yields the claim. □

Increasing the cutoff n keeps more structure of the PTRS, so the bounds get tighter.

Theorem 5 (Higher n Yields Tighter Bounds)**.** *For all $n \leq n'$, we have*

$$\sup_\sigma \mathbb{P}^{\max}(s\sigma \to_{\mathcal{P}}^* t\sigma) \quad \leq \quad \mathbb{P}^{\max}_{\mathcal{M}_{n'}(\mathcal{P})}(\lceil s \rceil_{n'}, T_t^{n'}) \quad \leq \quad \mathbb{P}^{\max}_{\mathcal{M}_n(\mathcal{P})}(\lceil s \rceil_n, T_t^n) \quad \leq \quad 1.$$

Proof Sketch. The first inequality is Thm. 4 (taking the supremum over all σ). For the second one, consider the map $t' \mapsto \lceil t' \rceil_n$ of $\mathcal{M}_{n'}(\mathcal{P})$ into $\mathcal{M}_n(\mathcal{P})$. Every narrowing step in $\mathcal{M}_{n'}(\mathcal{P})$ is matched by one in $\mathcal{M}_n(\mathcal{P})$, and $T_t^{n'}$ maps into T_t^n. So every scheduler of $\mathcal{M}_{n'}(\mathcal{P})$ corresponds to one scheduler of $\mathcal{M}_n(\mathcal{P})$ reaching the target with at least the same probability. □

Example 6. Reconsider $\mathcal{P}_{\mathsf{geo}}$ with $\mathsf{geo}(x) \to \{{}^1\!/_2\mathsf{geo}(\mathsf{s}(x)),\ {}^1\!/_2 x\}$ and the reachability problem $\mathbb{P}^{\max}(\mathsf{geo}(\mathsf{0}) \to_{\mathcal{P}}^* \mathsf{s}(\mathsf{0}))$, whose best upper bound is $1/4$. The part of $\mathcal{M}_2(\mathcal{P}_{\mathsf{geo}})$ that is reachable from the start term $\mathsf{geo}(\mathsf{0})$ is shown in Fig. 2. In $\mathsf{geo}(\mathsf{s}(x))$, the right branch $\mathsf{s}(x)$ unifies with $\lceil \mathsf{s}(\mathsf{0}) \rceil_2 = \mathsf{s}(\mathsf{0})$ and is thus a target, while the left branch returns to $\mathsf{geo}(\mathsf{s}(x))$ since the second s is cut off. This yields $\mathbb{P}^{\max}_{\mathcal{M}_2(\mathcal{P}_{\mathsf{geo}})}(\mathsf{geo}(\mathsf{0}), T^2_{\mathsf{s}(\mathsf{0})}) = 1/2$, proving $\mathsf{geo}(\mathsf{0}) \dashrightarrow^{\mathcal{P}_{\mathsf{geo}}}_{\leq 1/2} \mathsf{s}(\mathsf{0})$. The bound over-approximates $1/4$ because $\mathsf{geo}(\mathsf{s}(x))$ merges $\mathsf{geo}(\mathsf{s}(\mathsf{0}))$ with all deeper terms $\mathsf{geo}(\mathsf{s}^k(\mathsf{0}))$. Refining to $n = 3$ separates further terms, see Fig. 3. Now $\mathsf{geo}(\mathsf{s}(\mathsf{0}))$ and $\mathsf{geo}(\mathsf{s}(\mathsf{s}(x)))$ are distinct states, and only the former reaches the target $\mathsf{s}(\mathsf{0})$, while the latter loops and escapes to $\mathsf{s}(\mathsf{s}(x))$. Hence $\mathbb{P}^{\max}_{\mathcal{M}_3(\mathcal{P}_{\mathsf{geo}})}(\mathsf{geo}(\mathsf{0}), T^3_{\mathsf{s}(\mathsf{0})}) = 1/2 \cdot 1/2 = 1/4$, the exact value.

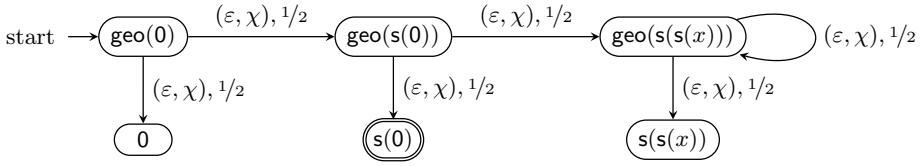


Figure 3: The reachable part of $\mathcal{M}_3(\mathcal{P}_{\mathsf{geo}})$ for $\mathcal{P}_{\mathsf{geo}}$. Here, $\mathsf{geo}(\mathsf{s}(\mathsf{0}))$ and $\mathsf{geo}(\mathsf{s}(\mathsf{s}(x)))$ are distinct states and only the former reaches the target $\mathsf{s}(\mathsf{0}) \in T^3_{\mathsf{s}(\mathsf{0})}$. Thus, $\mathbb{P}^{\max}_{\mathcal{M}_3(\mathcal{P}_{\mathsf{geo}})}(\mathsf{geo}(\mathsf{0}), T^3_{\mathsf{s}(\mathsf{0})}) = 1/4$. As before, by χ we denote the only rule in $\mathcal{P}_{\mathsf{geo}}$.

4 Upper Bounds on Reachability via Interpretations

Next, we lift the notion of *term orderings* [10] to the probabilistic setting. Similar ideas have been used for probabilistic imperative programs in, e.g., [2, 3]. Compared to the classical setting,

we have to use interpretations that allow the definition of an expected value and that induce a supermartingale. Then, we can obtain an upper bound on $\sup_\sigma \mathbb{P}^{\max}(s\sigma \to_{\mathcal{P}}^* t\sigma)$ instead of simple qualitative answers whether $s \to^* t$ holds or not. For simplicity we only consider weight functions instead of more complex term orderings as in [10]. A *weight function* is a mapping $W : \Sigma \to \mathbb{N}$ and extends to a function on terms $W : \mathcal{T}(\Sigma, \mathcal{V}) \to \mathbb{N}[\mathcal{V}]$ by defining $W(x) = x$ for all $x \in \mathcal{V}$, and $W(f(t_1, \dots, t_n)) = W(f) + W(t_1) + \cdots + W(t_n)$ otherwise. The relation $\leq$ on $\mathbb{N}[\mathcal{V}]$ is defined as usual: $p(x_1, \dots, x_n) \leq p'(x_1, \dots, x_n)$ if this holds for all instantiations of $x_1, \dots x_n$ with natural numbers. The expected weight of a finite multi-distribution $\mu = \{{}^{p_1}t_1, \dots, {}^{p_k}t_k\}$ on terms is defined as $\mathbb{E}_W(\mu) = \sum_{1 \leq i \leq k} p_i \cdot W(t_i)$. For example, consider the PTRS $\mathcal{P}_{\mathsf{rw}}$ with a unary function symbol s and a constant symbol $\mathsf{0}$ with the only rules $\mathsf{s}(x) \to \{{}^{1/2}x, {}^{1/2}\mathsf{s}^2(x)\}$, which describes a one dimensional symmetric *random walk* over natural numbers in Peano-notation. A weight function $W : \{\mathsf{s}, \mathsf{0}\} \to \mathbb{N}$ is given by $W(\mathsf{0}) = 0$ and $W(\mathsf{s}) = 1$. Then $W(\mathsf{s}(x)) = x + 1$ and $\mathbb{E}_W(\{{}^{1/2}x, {}^{1/2}\mathsf{s}^2(x)\}) = 1/2 \cdot x + 1/2 \cdot (x + 2) = x + 1$ as well.

We now sketch how a weight function W that induces a supermartingale can be used to infer an upper bound on $\sup_\sigma \mathbb{P}^{\max}(s\sigma \to_{\mathcal{P}}^* t\sigma)$. The main idea is to approximate all rewrite sequences starting in $s\sigma$ by a supermartingale that tracks the weight of the current term. To this end, we require that W does not increase in expectation along rewrite steps, i.e.,

$$W(\ell) \geq \mathbb{E}_W(\mu) \qquad \text{for all rules } \ell \to \mu \in \mathcal{P}. \tag{1}$$

The inequation (1) lifts from rules to rewrite steps: whenever $t \to_{\mathcal{P}} \mu$, we have $W(t) \geq \mathbb{E}_W(\mu)$. Intuitively, this means that with every rewrite step we either move away from the target or stay at the same distance in expectation, i.e., we have

$$\begin{array}{ccccccccccc} & \{{}^1 s\} & \to_{\mathcal{P}} & \mu_1 & \to_{\mathcal{P}} & \mu_2 & \to_{\mathcal{P}} & \mu_3 & \to_{\mathcal{P}} & \cdots \\ \implies & \mathbb{E}_W(\{{}^1 s\}) & \geq & \mathbb{E}_W(\mu_1) & \geq & \mathbb{E}_W(\mu_2) & \geq & \mathbb{E}_W(\mu_3) & \geq & \cdots \end{array}$$

If we have $W(t) > W(s)$, then we can bound the reachability probability $\sup_\sigma \mathbb{P}^{\max}(s\sigma \to_{\mathcal{P}}^* t\sigma)$: we cannot move with probability 1 from s to t, since this would imply that the expected value has to increase eventually. The Stopping Theorem for supermartingales yields a concrete bound.

We now give a formal definition of the induced supermartingale. Let σ be a ground substitution for s and t, and let $\mathfrak{T}$ be a $\to$-RT with $\mathrm{root}(\mathfrak{T}) = s\sigma$. We extend W to nodes by $W(v) = W(a_v)$ and to a fresh symbol $\bot \notin V(\mathfrak{T})$ by $W(\bot) = 0$. We turn $\mathfrak{T}$ into a stochastic process $\{X_n\}_{n \in \mathbb{N}}$ over $V(\mathfrak{T}) \uplus \{\bot\}$: (1) X_0 is the root of $\mathfrak{T}$; (2) if $X_n = v$ has children $w_1, \dots, w_k$, then $X_{n+1} = w_i$ with probability ${}^{p_{w_i}}/_{p_v}$. By definition of RTs, $\{{}^{p_{w_1}/p_v}a_{w_1}, \dots, {}^{p_{w_k}/p_v}a_{w_k}\}$ is exactly the distribution μ of the rewrite step $a_v \to \mu$; and (3) if $X_n = v$ is a leaf or $X_n = \bot$, then $X_{n+1} = \bot$ with probability 1. Thus, the realizations of $\{X_n\}_{n \in \mathbb{N}}$ are exactly the maximal paths of $\mathfrak{T}$, weighted by their probabilities. Let $Y_n = W(X_n)$ track the weight along the path. Then $\{Y_n\}_{n \in \mathbb{N}}$ is non-negative, and a *supermartingale*: the expected weight of the successor X_{n+1} never exceeds the current weight Y_n.[1] Indeed, for an inner node $X_n = v$ with $a_v \to \mu$, this expected weight is $\mathbb{E}_W(\mu) \leq W(a_v) = Y_n$ by inequation (1), and for a leaf or $\bot$ it is $0 \leq Y_n$.

To capture the event of reaching a term of weight at least $W(t\sigma)$, we use the *stopping time* $N = \inf\{\, n \in \mathbb{N} \mid Y_n \geq W(t\sigma) \,\} \in \mathbb{N} \cup \{\infty\}$ (with $\inf \varnothing = \infty$), i.e., we stop once we see a weight greater than or equal to $W(t\sigma)$. Since the event $\{Y_n \geq W(t\sigma)\}$ depends only on $X_0, \dots, X_n$, N is indeed a stopping time. Intuitively, we run the process until the current term has weight at least $W(t\sigma)$, and run forever otherwise. Now the following stopping theorem yields $\mathbb{E}(Y_0) \geq \mathbb{E}(Y_N)$ and thereby our upper bound on $\mathbb{P}^{\max}(s\sigma \to_{\mathcal{P}}^* t\sigma)$.

[1] Formally, $\mathbb{E}(Y_{n+1} \mid \mathfrak{F}_n) \leq Y_n$ for the natural filtration $\{\mathfrak{F}_n\}_{n \in \mathbb{N}}$ of $\{X_n\}_{n \in \mathbb{N}}$.

Theorem 7 (Stopping Theorem (Theorem 4.8.4 in [4]))**.** *Let $\{Y_n\}_{n\in\mathbb{N}}$ be a non-negative supermartingale and $N \in \mathbb{N} \cup \{\infty\}$ a stopping time. Then $\mathbb{E}(Y_0) \geq \mathbb{E}(Y_N)$.*

Corollary 8. *Let s, t be terms in a PTRS $\mathcal{P}$, W be a weight function with $W(\ell) \geq \mathbb{E}_W(\mu)$ for all rules $\ell \to \mu \in \mathcal{P}$, and σ be a ground substitution for s and t. If $W(t\sigma) \geq W(s\sigma)$ and $W(t\sigma) > 0$ then $\mathbb{P}^{\max}(s\sigma \to^*_{\mathcal{P}} t\sigma) \leq W(s\sigma) \cdot W(t\sigma)^{-1}$.*

Proof Sketch. We instantiate the construction above for some $\to$-RT $\mathfrak{T}$ with $\mathrm{root}(\mathfrak{T}) = s\sigma$, yielding the non-negative supermartingale $\{Y_n\}_{n\in\mathbb{N}}$ and the stopping time N. By construction, N is the first time the process hits a node ${}^{p_N}a_N$ in $\mathfrak{T}$ where $W(a_N) \geq W(t\sigma)$. Using Thm. 7 we can infer that $\mathbb{E}(Y_0) \geq \mathbb{E}(Y_N)$. Note that our process always starts in $Y_0 = W(s\sigma)$. Moreover, $p^* := \mathbb{P}[N < \infty]$ is the probability that a path in $\mathfrak{T}$ ever hits a node ${}^{p_N}a_N$ with $W(a_N) \geq W(t\sigma)$. Since every path that reaches a node labeled with $t\sigma$ hits a node of weight $W(t\sigma)$ there, we have $\mathbb{P}_{\mathfrak{T}}(s\sigma \to^* t\sigma) \leq p^*$.

Finally, recall that $Y_N \geq W(t\sigma)$ with probability p^* and $Y_N \geq 0$ otherwise. Hence, $\mathbb{E}(Y_N) \geq p^* \cdot W(t\sigma) + (1 - p^*) \cdot 0 = p^* \cdot W(t\sigma)$, and together with $\mathbb{E}(Y_0) \geq \mathbb{E}(Y_N)$ from above we obtain $W(s\sigma) \geq p^* \cdot W(t\sigma)$. So we can conclude that $\mathbb{P}_{\mathfrak{T}}(s\sigma \to^* t\sigma) \leq p^* \leq W(s\sigma) \cdot W(t\sigma)^{-1}$. Note that we can obtain the same upper bound for all $\to$-RTs $\mathfrak{T}$ with root ${}^1s\sigma$. Hence, we can ultimately conclude that $\mathbb{P}^{\max}(s\sigma \to^*_{\mathcal{P}} t\sigma) \leq W(s\sigma) \cdot W(t\sigma)^{-1}$ which proves the claim. □

Now we can combine all results so far to obtain the main result of this section.

Theorem 9 (Upper Bounds on Max Reachability via Weight Functions)**.** *Let s, t be terms in a PTRS $\mathcal{P}$, and W be a weight function with $W(\ell) \geq \mathbb{E}_W(\mu)$ for all rules $\ell \to \mu \in \mathcal{P}$ such that $W(t) > 0$. Then $\sup_\sigma \mathbb{P}^{\max}(s\sigma \to^*_{\mathcal{P}} t\sigma) \leq p$ for*

$$p = \begin{cases} \sup_{\alpha\colon \mathcal{V}\to\mathbb{N}} \big(W(s)[v/\alpha(v) \mid v \in \mathcal{V}] \cdot (W(t)[v/\alpha(v) \mid v \in \mathcal{V}])^{-1}\big), & W(t\sigma) \geq W(s\sigma) \text{ for all } \sigma, \\ 1, & \text{otherwise.} \end{cases}$$

Proof Sketch. If $W(t\sigma) < W(s\sigma)$ for some ground substitution σ, then $p = 1$ is a trivial upper bound and the claim is immediate. Otherwise, $W(t\sigma) \geq W(s\sigma)$ holds for all ground substitutions σ, and Cor. 8 yields $\mathbb{P}^{\max}(s\sigma \to^*_{\mathcal{P}} t\sigma) \leq W(s\sigma) \cdot W(t\sigma)^{-1}$ for each such σ. Finally note that $\sup_{\alpha\colon \mathcal{V}\to\mathbb{N}} \big(W(s)[v/\alpha(v) \mid v \in \mathcal{V}] \cdot (W(t)[v/\alpha(v) \mid v \in \mathcal{V}])^{-1}\big) \geq \sup_\sigma (W(s\sigma) \cdot W(t\sigma)^{-1})$, since $W(u\sigma) = W(u)[v/W(v\sigma) \mid v \in \mathcal{V}]$ and α ranges freely over $\mathcal{V} \to \mathbb{N}$, while $W(v\sigma)$ ranges only over a subset of $\mathbb{N}$. Hence $\mathbb{P}^{\max}(s\sigma \to^*_{\mathcal{P}} t\sigma) \leq p$ for all σ, which proves the claim. □

Example 10. Consider the PTRS $\mathcal{P}_{\mathsf{rw}}$ with the only rules $\mathsf{s}(x) \to \{{}^1/_2x, {}^1/_2\mathsf{s}^2(x)\}$ again. Using the weight function $W(\mathsf{0}) = 0$ and $W(\mathsf{s}) = 1$, we get $W(\mathsf{s}(x)) = x + 1 = {}^1/_2 \cdot x + {}^1/_2 \cdot (x + 2) = \mathbb{E}_W(\{{}^1/_2x, {}^1/_2\mathsf{s}^2(x)\})$. Therefore, we can infer $\mathsf{s}(\mathsf{0}) \dashrightarrow^{\mathcal{P}_{\mathsf{rw}}}_{\leq {}^1/_3} \mathsf{s}^3(\mathsf{0})$, since $W(\mathsf{s}(\mathsf{0})) = 1$, $W(\mathsf{s}^3(\mathsf{0})) = 3$, and $W(s) \cdot W(t)^{-1} = {}^1/_3$.

In the future, we will investigate the relation between both presented techniques.

Acknowledgements We thank Éléanore Meyer for feedback on an earlier version, and the anonymous reviewers for their comments.

References

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

[2] Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, and Đorđe Žikelić. Sound and Complete Certificates for Quantitative Termination Analysis of Probabilistic Programs. In *Proc. CAV '22*, LNCS 13371, pages 55–78, 2022.

[3] Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. Stochastic invariants for probabilistic termination. In *Proc. POPL '17*, ?, pages 145–160, 2017.

[4] Rick Durrett. *Probability: Theory and Examples*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 5 edition, 2019.

[5] 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. *Journal of Automated Reasoning*, 58(1):3–31, 2017.

[6] Raúl Gutiérrez and Salvador Lucas. Automatically Proving and Disproving Feasibility Conditions. In *Proc. IJCAR '20*, LNCS 12167, pages 416–435, 2020.

[7] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. The Probabilistic Model Checker Storm. *International Journal on Software Tools for Technology Transfer*, 24:589–610, 2022.

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

[9] Christian Sternagel and Akihisa Yamada. Reachability Analysis for Termination and Confluence of Rewriting. In *Proc. TACAS '19*, LNCS 11429, pages 262–278, 2019.

[10] Akihisa Yamada. Term Orderings for Non-reachability of (Conditional) Rewriting. In *Proc. IJCAR '22*, LNCS 13385, pages 248–267, 2022.

Enumerating Ground Canonical Rewrite Systems

Masahiko Sakai[1], Aart Middeldorp[2], and Sarah Winkler[3]

[1] Nagoya University, Nagoya, Japan, sakai@rewriting.jp
[2] University of Innsbruck, Innsbruck, Austria, aart.middeldorp@uibk.ac.at
[3] Free University of Bozen-Bolzano, Bolzano, Italy, winkler@inf.unibz.it

Abstract

In an earlier paper we proved that a transformation due to Snyder generates all canonical TRSs equivalent to a given canonical ground TRS. Here we present an explicit recursive procedure to generate these. We prove its correctness and show how the procedure can be used to obtain the exponential upper bound due to Snyder on the number of canonical ground presentations.

1 Introduction

Congruence closure is an efficient technique to solve the word problem for systems of ground equations. Completion is a well-known technique for transforming systems of equations into equivalent canonical rewrite systems. A rewrite system is canonical if it confluent, terminating and *reduced*. Completion takes a reduction order as input and if it succeeds, the word problem is decidable by computing and comparing the unique normal forms of the two terms involved. For systems of ground equations, completion can be tamed such that it always terminates. Snyder [3, Theorem 4.7] shows that the number of distinct canonical rewrite systems representing a given set of ground equations is at most 2^k where k is the number of equations. Furthermore, each of these canonical rewrite systems has the same number of rewrite rules. Finally, given one canonical rewrite system, all others can be obtained by a simple transformation. This transformation is denoted by $(\star)$ and defined as follows:

> Let $\mathcal{R} \uplus \{\ell \to r\}$ be a canonical ground TRS such that r is not a proper subterm of ℓ. Let $\mathcal{R}'$ be the ground TRS obtained from $\mathcal{R}$ by replacing every occurrence of r in the rewrite rules by ℓ. The result of the transformation is given by $\mathcal{R}' \cup \{r \to \ell\}$.

Given ground TRSs $\mathcal{R}$ and $\mathcal{S}$, we write $\mathcal{R} \Longrightarrow \mathcal{S}$ if $\mathcal{S}$ can be obtained from $\mathcal{R}$ by an application of the transformation $(\star)$.

Example 1. The collection of ground equations

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

admits three different canonical TRSs, which are connected by $(\star)$ as follows:

$$\begin{array}{rl} \mathsf{f}(\mathsf{g}(\mathsf{b},\mathsf{b})) \xrightarrow{1} \mathsf{g}(\mathsf{b},\mathsf{b}) \\ \mathsf{a} \xrightarrow{2} \mathsf{g}(\mathsf{b},\mathsf{b}) \\ \mathsf{h}(\mathsf{g}(\mathsf{b},\mathsf{b})) \xrightarrow{3} \mathsf{b} \end{array} \quad \overset{2}{\Longleftrightarrow} \quad \begin{array}{rl} \mathsf{f}(\mathsf{a}) \xrightarrow{1} \mathsf{a} \\ \mathsf{g}(\mathsf{b},\mathsf{b}) \xrightarrow{2} \mathsf{a} \\ \mathsf{h}(\mathsf{a}) \xrightarrow{3} \mathsf{b} \end{array} \quad \overset{3}{\Longleftrightarrow} \quad \begin{array}{rl} \mathsf{f}(\mathsf{a}) \xrightarrow{1} \mathsf{a} \\ \mathsf{g}(\mathsf{h}(\mathsf{a}),\mathsf{h}(\mathsf{a})) \xrightarrow{2} \mathsf{a} \\ \mathsf{b} \xrightarrow{3} \mathsf{h}(\mathsf{a}) \end{array}$$

Here the number above the $\Longleftrightarrow$ arrow indicates the rule to which transformation $(\star)$ was applied. Detailed proofs concerning the transformation $(\star)$ are presented in our earlier paper [2]. We also showed that it fails in the AC case. The following statement expresses the crucial soundness property of the transformation [2, Lemma 4].

Lemma 1. *The* TRS $\mathcal{R}' \cup \{r \to \ell\}$ *is canonical.*

In this note we present an explicit procedure to generate all canonical presentations equivalent to a given ground canonical TRS. The correctness of the procedure is based on the completeness of ($\star$). The procedure is used to give a simple proof of the exponential upper bound on the number of canonical ground presentations.

We assume familiarity with term rewriting [1]. Given a TRS $\mathcal{R}$, we denote the set of left-hand (right-hand) sides of its rules by $\mathsf{LHS}(\mathcal{R})$ ($\mathsf{RHS}(\mathcal{R})$). We write $t\{\ell/r\}$ for the term obtained from t after replacing all subterms r by ℓ. Similarly, $\mathcal{R}\{\ell/r\}$ denotes the TRS $\{s\{\ell/r\} \to t\{\ell/r\} \mid s \to t \in \mathcal{R}\}$. We write $\trianglelefteq$ for the subterm relation. Two TRSs $\mathcal{R}$ and $\mathcal{S}$ are *(conversion) equivalent* if $\leftrightarrow^*_{\mathcal{R}} = \leftrightarrow^*_{\mathcal{S}}$. We denote the set of canonical presentations of a ground TRS $\mathcal{R}$ by $\mathsf{CaP}(\mathcal{R})$ and abbreviate

$$\prod_{v \in \mathsf{RHS}(\mathcal{R})} \bigl(1 + \#\{\ell \to r \in \mathcal{R} \mid r = v\}\bigr)$$

by $\mathsf{bound}(\mathcal{R})$. The following result will be heavily used in the sequel. (A TRS $\mathcal{R}$ is reduced if $\ell \in \mathsf{NF}(\mathcal{R} \setminus \{\ell \to r\})$ and $r \in \mathsf{NF}(\mathcal{R})$ for every rule $\ell \to r$ of $\mathcal{R}$.)

Theorem 1 (Snyder [3, Theorem 2.14])**.** *Reduced ground* TRS*s are canonical.* □

2 Enumeration

Let T be a finite set of ground terms T. If $t \in T$ then we write T_{-t} for $T \setminus \{t\}$ and $\mathcal{T}_{\to t}$ denotes the TRS $\{s \to t \mid s \in T_{-t}\}$.

Definition 1. Given a ground canonical TRS $\mathcal{R}$, we define the set $E(\mathcal{R})$ recursively as follows: If $\mathcal{R} = \varnothing$ then $E(\mathcal{R}) = \{\varnothing\}$. If $\mathcal{R} \neq \varnothing$ then we choose a right-hand side $t \in \mathsf{RHS}(\mathcal{R})$ and let $T = \{t\} \cup \{s \mid s \to t \in \mathcal{R}\}$ and $\mathcal{S} = \mathcal{R} \setminus \mathcal{T}_{\to t}$ in

$$E(\mathcal{R}) = \left\{ (\mathcal{U} \cup \mathcal{V}_{\to v})\{v/\bullet_t\} \;\middle|\; \begin{array}{l} \mathcal{U} \in E(\mathcal{S}\{\bullet_t/t\}),\ V = (\{t\} \cup T_{-t}\{\bullet_t/t\})\!\downarrow_{\mathcal{U}} \\ \text{and } v \in V \text{ such that } \bullet_t \notin v \end{array} \right\}$$

Here $\bullet_t$ is a fresh constant whose purpose is to protect the term t from being modified in the recursive call to E.

The choice of $t \in \mathsf{RHS}(\mathcal{R})$ in the definition of $E(\mathcal{R})$ does not affect the outcome. This follows from the proofs below.

We write $\bullet_t \in s$ for a term s to denote that $\bullet_t$ is a subterm of s, $\bullet_t \in T$ for a set of terms T if $\bullet_t$ is an element of T, and $\bullet_t \in \mathcal{R}$ for a TRS $\mathcal{R}$ if $\bullet_t$ is a left- or right-hand side of $\mathcal{R}$. To simplify the notation, we write $\mathcal{R}\langle\bullet_t\rangle$ for $\mathcal{R}\{\bullet_t/t\}$ in the sequel. Before proving that $E(\mathcal{R})$ consists of all canonical presentations of $\mathcal{R}$, we illustrate the definition on a concrete example.

Example 2. Consider the TRS $\mathcal{R}$ consisting of the three rules

$$\mathsf{f}(\mathsf{g}(\mathsf{b},\mathsf{b})) \xrightarrow{1} \mathsf{g}(\mathsf{b},\mathsf{b}) \qquad \mathsf{a} \xrightarrow{2} \mathsf{g}(\mathsf{b},\mathsf{b}) \qquad \mathsf{h}(\mathsf{g}(\mathsf{b},\mathsf{b})) \xrightarrow{3} \mathsf{b}$$

We select $t = \mathsf{g}(\mathsf{b},\mathsf{b}) \in \mathsf{RHS}(\mathcal{R})$ and obtain $T = \{t, \mathsf{a}, \mathsf{f}(\mathsf{g}(\mathsf{b},\mathsf{b}))\}$ and $\mathcal{S} = \{\mathsf{h}(\mathsf{g}(\mathsf{b},\mathsf{b})) \to \mathsf{b}\}$. Computing $E(\mathcal{S}\langle\bullet_t\rangle) = E(\{\mathsf{h}(\bullet_t) \to \mathsf{b}\})$ yields $\mathcal{U}_1 = \{\mathsf{h}(\bullet_t) \to \mathsf{b}\}$ and $\mathcal{U}_2 = \{\mathsf{b} \to \mathsf{h}(\bullet_t)\}$.

- For $\mathcal{U}_1$ we obtain $V = \{t, \mathsf{a}, \mathsf{f}(\bullet_t)\}\downarrow_{\mathcal{U}_1} = \{\mathsf{g}(\mathsf{b},\mathsf{b}), \mathsf{a}, \mathsf{f}(\bullet_t)\}$. There are two choices for v: $\mathsf{g}(\mathsf{b},\mathsf{b})$ and a, resulting in the respective TRSs

$$\begin{aligned} \mathsf{f}(\mathsf{g}(\mathsf{b},\mathsf{b})) &\to \mathsf{g}(\mathsf{b},\mathsf{b}) \\ \mathsf{a} &\to \mathsf{g}(\mathsf{b},\mathsf{b}) \\ \mathsf{h}(\mathsf{g}(\mathsf{b},\mathsf{b})) &\to \mathsf{b} \end{aligned} \qquad \text{and} \qquad \begin{aligned} \mathsf{f}(\mathsf{a}) &\to \mathsf{a} \\ \mathsf{g}(\mathsf{b},\mathsf{b}) &\to \mathsf{a} \\ \mathsf{h}(\mathsf{a}) &\to \mathsf{b} \end{aligned}$$

- For $\mathcal{U}_2$ we obtain $V = \{t, \mathsf{a}, \mathsf{f}(\bullet_t)\}\downarrow_{\mathcal{U}_2} = \{\mathsf{g}(\mathsf{h}(\bullet_t), \mathsf{h}(\bullet_t)), \mathsf{a}, \mathsf{f}(\bullet_t)\}$. Because of the condition $\bullet_t \notin v$, there is only one choice for v: a, resulting in the TRS

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

Next, we show that the choice of t does not affect the result; suppose $t = \mathsf{b}$. In that case we obtain $T = \{t, \mathsf{h}(\mathsf{g}(\mathsf{b},\mathsf{b}))\}$ and $\mathcal{S} = \{\mathsf{f}(\mathsf{g}(\mathsf{b},\mathsf{b})) \to \mathsf{g}(\mathsf{b},\mathsf{b}), \mathsf{a} \to \mathsf{g}(\mathsf{b},\mathsf{b})\}$. Computing $E(\mathcal{S}\langle\bullet_t\rangle)$ produces the two TRSs $\mathcal{U}_1 = \{\mathsf{f}(\mathsf{g}(\bullet_t,\bullet_t)) \to \mathsf{g}(\bullet_t,\bullet_t), \mathsf{a} \to \mathsf{g}(\bullet_t,\bullet_t)\}$ and $\mathcal{U}_2 = \{\mathsf{f}(\mathsf{a}) \to \mathsf{a}, \mathsf{g}(\bullet_t,\bullet_t) \to \mathsf{a}\}$.

- For $\mathcal{U}_1$ we obtain $V = \{t, \mathsf{h}(\mathsf{g}(\bullet_t,\bullet_t))\}\downarrow_{\mathcal{U}_1} = \{\mathsf{b}, \mathsf{h}(\mathsf{g}(\bullet_t,\bullet_t))\}$. The only choice for v is b, resulting in the TRS

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

- For $\mathcal{U}_2$ we obtain $V = \{t, \mathsf{h}(\mathsf{g}(\bullet_t,\bullet_t))\}\downarrow_{\mathcal{U}_2} = \{\mathsf{b}, \mathsf{h}(\mathsf{a})\}$. There are two choices for v: b and $\mathsf{h}(\mathsf{a})$, resulting in the respective TRSs

$$\begin{aligned} \mathsf{f}(\mathsf{a}) &\to \mathsf{a} \\ \mathsf{g}(\mathsf{b},\mathsf{b}) &\to \mathsf{a} \\ \mathsf{h}(\mathsf{a}) &\to \mathsf{b} \end{aligned} \qquad \text{and} \qquad \begin{aligned} \mathsf{f}(\mathsf{a}) &\to \mathsf{a} \\ \mathsf{g}(\mathsf{h}(\mathsf{a}),\mathsf{h}(\mathsf{a})) &\to \mathsf{a} \\ \mathsf{b} &\to \mathsf{h}(\mathsf{a}) \end{aligned}$$

3 Correctness

In this section we prove that the recursive procedure of Definition 1 generates all canonical presentations of a given ground canonical TRS. We write $T \leftrightarrow^*_{\mathcal{R}} U$ for a TRS $\mathcal{R}$ and sets of terms T and U if for any term $t \in T$ ($t \in U$) there is a term $u \in U$ ($u \in T$) such that $t \leftrightarrow^*_{\mathcal{R}} u$.

Definition 2. Given a ground TRS $\mathcal{R}$, a finite set T of ground terms, and a constant $\bullet$, we write $P(\mathcal{R}, T, \bullet)$ for the conjunction of the following conditions:

1. $\mathcal{R}$ is canonical,
2. $T \subseteq \mathsf{NF}(\mathcal{R})$,
3. no $t \in T$ is a subterm of a term in $\mathsf{LHS}(\mathcal{R}) \cup \mathsf{RHS}(\mathcal{R}) \cup T_{-t}$, and
4. $\bullet \notin \mathsf{LHS}(\mathcal{R}) \cup \mathsf{RHS}(\mathcal{R}) \cup T$.

Lemma 2. *If $P(\mathcal{R}, T, \bullet)$ and $t \in T$ with $\bullet \notin t$ then $\mathcal{R} \cup \mathcal{T}_{\to t}$ and $(\mathcal{R} \cup \mathcal{T}_{\to t})\{t/\bullet\}$ are canonical.*

Proof. Assume that $\ell \to r$ in the system $\mathcal{R} \cup \mathcal{T}_{\to t}$ reduces a term in $\mathcal{R} \cup \mathcal{T}_{\to t}$ different from ℓ. If $\ell \to r \in \mathcal{R}$ this contradicts conditions 1 or 2. Similarly, $\ell \to r \in \mathcal{T}_t$ causes a contradiction to condition 3. Combined with condition 4, the canonicity of $(\mathcal{R} \cup \mathcal{T}_{\to t})\{t/\bullet\}$ follows. □

Lemma 3. *Let $\mathcal{R}$ be a canonical ground* TRS. *If $\mathcal{R} \Longrightarrow^* \mathcal{S}$ and $P(\mathcal{R}, T, \bullet)$ then $P(\mathcal{S}, T\downarrow_{\mathcal{S}}, \bullet)$.*

Proof. It suffices to prove the statement for a single step $\mathcal{R} \Longrightarrow \mathcal{S}$. The TRS $\mathcal{S}$ is canonical as a consequence of Lemma 1. Because the terms in $T{\downarrow_{\mathcal{S}}}$ are normalized by $\mathcal{S}$, $T{\downarrow_{\mathcal{S}}}$ obviously consists of normal forms of $\mathcal{S}$. Let $\mathcal{R} = \mathcal{R}' \uplus \{\ell \to r\}$ and $\mathcal{S} = \mathcal{R}'\{\ell/r\} \cup \{r \to \ell\}$. Since ℓ and r are different from $\bullet$ by condition 4 of $P(\mathcal{R}, T, \bullet)$, also condition 4 of $P(\mathcal{S}, T{\downarrow_{\mathcal{S}}}, \bullet)$ holds. It remains to check the third condition. So let $t \in T{\downarrow_{\mathcal{S}}}$. Since $T \subseteq \mathsf{NF}(\mathcal{R})$, $t = t'\{\ell/r\}$ for some $t' \in T$. Suppose for a proof by contradiction that t is a subterm of a term in $\mathsf{LHS}(\mathcal{S}) \cup \mathsf{RHS}(\mathcal{S}) \cup T{\downarrow_{\mathcal{S}}} \setminus \{t\}$. First, we note that $t \not\trianglelefteq \ell$ (1): If $t \trianglelefteq \ell$ and $t = t'$ this contradicts condition 3. On the other hand, if $t \neq t'$ then t must contain ℓ, but $t = \ell$ is impossible because then $t' = r$, again contradicting condition 3. Second, $t \not\trianglelefteq r$ (2): If $t = t'$ this is excluded by condition 3, and if $t \neq t'$ then t must contain ℓ, but $\ell \trianglelefteq r$ contradicts termination of $\mathcal{R}$.

Now suppose t is a subterm of a term in $\mathsf{LHS}(\mathcal{S}) = \mathsf{LHS}(\mathcal{R}')\{\ell/r\} \cup \{r\}$. By (2), we must have that t is a subterm of a term $u \in \mathsf{LHS}(\mathcal{S})$ such that $u = u'\{\ell/r\}$ for some $u' \in \mathsf{LHS}(\mathcal{R}')$. Then t is a subterm of ℓ or $t' \trianglelefteq u'$, contradicting (1) and condition 3. Next, suppose t is a subterm of a term in $\mathsf{RHS}(\mathcal{S}) = \mathsf{RHS}(\mathcal{R}')\{\ell/r\} \cup \{\ell\}$. By (1), t must be a subterm of a term $v \in \mathsf{RHS}(\mathcal{S})$ such that $v = v'\{\ell/r\}$ for some $v' \in \mathsf{RHS}(\mathcal{R}')$. Then either $t \trianglelefteq \ell$ or $t' \trianglelefteq v'$, contradicting (1) and condition 3. Finally, if t is a subterm of a term $u \in T{\downarrow_{\mathcal{S}}} \setminus \{t\}$, there must be some $u' \in T$ such that $u = u'\{\ell/r\}$. Again either $t \trianglelefteq \ell$ or t must already be a subterm of u', contradicting (1) or condition 3. □

The following lemma collects some easy observations.

Lemma 4.

1. *Any subset of a ground canonical* TRS *is canonical.*
2. *If T is a set of ground terms then $t \leftrightarrow^*_{\mathcal{T}_{\to u}} v$ for all terms $t, u, v \in T$.*
3. *If $u \trianglelefteq v$ then $u\{t/\bullet_t\} \trianglelefteq v\{t/\bullet_t\}$. The converse holds if $t \not\trianglelefteq u$ and $t \not\trianglelefteq v$.*

Lemma 5. *If $\mathcal{R} \Longrightarrow^* \mathcal{S}$, $\bullet \notin t \in T$ and $\bullet \notin u \in U = T{\downarrow_{\mathcal{S}}}$ then the* TRS*s $(\mathcal{R} \cup \mathcal{T}_{\to t})\{t/\bullet\}$ and $(\mathcal{S} \cup \mathcal{U}_{\to u})\{u/\bullet\}$ are equivalent.*

Proof. We obtain $\leftrightarrow^*_{\mathcal{R}} = \leftrightarrow^*_{\mathcal{S}}$ (1) from $\mathcal{R} \Longrightarrow^* \mathcal{S}$. Let $\mathcal{R}' = (\mathcal{R} \cup \mathcal{T}_{\to t})\{t/\bullet\}$ and $\mathcal{S}' = (\mathcal{S} \cup \mathcal{U}_{\to u})\{u/\bullet\}$. We show $\leftrightarrow^*_{\mathcal{R}'} = \leftrightarrow^*_{\mathcal{S}'}$ below.

$\subseteq$ We have $t \to^*_{\mathcal{S}'} t{\downarrow_{\mathcal{S}'}} \leftrightarrow^*_{\mathcal{S}'} u$. Since $\bullet \notin t$ and $\bullet \notin u$, $t = t\{u/\bullet\} \leftrightarrow^*_{\mathcal{S}'} u\{u/\bullet\} = u$ (2). Let $\ell' \to r' \in \mathcal{R}'$. We show $\ell' \leftrightarrow^*_{\mathcal{S}'} r'$ by distinguishing two cases. If $\ell' \to r' \in \mathcal{R}\{t/\bullet\}$ then $\ell' = \ell\{t/\bullet\}$ and $r' = r\{t/\bullet\}$ for some rule $\ell \to r \in \mathcal{R}$. We obtain $\ell \leftrightarrow^*_{\mathcal{S}} r$ from (1). Hence $\ell' \leftrightarrow^*_{\mathcal{S}'} \ell\{u/\bullet\} \leftrightarrow^*_{\mathcal{S}'} r\{u/\bullet\} \leftrightarrow^*_{\mathcal{S}'} r'$ using (2). Next consider $\ell' \to r' \in \mathcal{T}_{\to t}\{t/\bullet\}$. We have

$$U\{u/\bullet\} = T{\downarrow_{\mathcal{S}}}\{u/\bullet\} = T\{u/\bullet\}{\downarrow_{\mathcal{S}\{u/\bullet\}}} \leftrightarrow^*_{\mathcal{S}\{u/\bullet\}} T\{u/\bullet\} \leftrightarrow^*_{\mathcal{S}'} T\{t/\bullet\}$$

using (2) and because $u \in \mathsf{NF}(\mathcal{S})$. So $U\{u/\bullet\} \leftrightarrow^*_{\mathcal{S}'} T\{t/\bullet\}$ and since $\ell', r' \in T\{t/\bullet\}$ there exist terms $\ell, r \in U\{u/\bullet\}$ such that $\ell' \leftrightarrow^*_{\mathcal{S}'} \ell$ and $r' \leftrightarrow^*_{\mathcal{S}'} r$. From $\bullet \notin u$ we infer $u \in U\{u/\bullet\}$ and $(U\{u/\bullet\})_u = U_u\{u/\bullet\}$. Hence $\ell \leftrightarrow^*_{U_u\{u/\bullet\}} r$ by Lemma 4(2) and thus $\ell' \leftrightarrow^*_{\mathcal{S}'} \ell \leftrightarrow^*_{\mathcal{S}'} r \leftrightarrow^*_{\mathcal{S}'} r'$ as desired.

$\supseteq$ Since $u \in U = T{\downarrow_{\mathcal{S}}}$ there exists a term $t' \in T$ such that $t'{\downarrow_{\mathcal{S}}} = u$. Using (1) and Lemma 4(2) we obtain $t \leftrightarrow^*_{\mathcal{T}_{\to t}} t' \leftrightarrow^*_{\mathcal{R}} u$. Since $\bullet \notin t$ and $\bullet \notin u$, $t = t\{t/\bullet\} \leftrightarrow^*_{\mathcal{R}'} u\{t/\bullet\} = u$ (3). Let $\ell' \to r' \in \mathcal{S}'$. We show $\ell' \leftrightarrow^*_{\mathcal{R}'} r'$ by distinguishing two cases. If $\ell' \to r' \in \mathcal{S}\{u/\bullet\}$ then $\ell' = \ell\{u/\bullet\}$ and $r' = r\{u/\bullet\}$ for some rule $\ell \to r \in \mathcal{S}$. We obtain $\ell \leftrightarrow^*_{\mathcal{R}} r$ from (1). Hence $\ell' \leftrightarrow^*_{\mathcal{R}'} \ell\{t/\bullet\} \leftrightarrow^*_{\mathcal{R}'} r\{t/\bullet\} \leftrightarrow^*_{\mathcal{R}'} r'$ using (3). Next consider $\ell' \to r' \in \mathcal{U}_{\to u}\{u/\bullet\}$. We have $T \leftrightarrow^*_{\mathcal{S}} T{\downarrow_{\mathcal{S}}}$ and thus $T \leftrightarrow^*_{\mathcal{R}} T{\downarrow_{\mathcal{S}}}$ by (1). Hence

$$U\{u/\bullet\} \leftrightarrow^*_{\mathcal{R}'} U\{t/\bullet\} = T{\downarrow_{\mathcal{S}}}\{t/\bullet\} \leftrightarrow^*_{\mathcal{R}\{t/\bullet\}} T\{t/\bullet\}$$

using (3). So $U\{u/\bullet\} \leftrightarrow^*_{\mathcal{R}'} T\{t/\bullet\}$ and since $\ell', r' \in U\{u/\bullet\}$ there exist terms $\ell, r \in T\{t/\bullet\}$ such that $\ell' \leftrightarrow^*_{\mathcal{R}'} \ell$ and $r' \leftrightarrow^*_{\mathcal{R}'} r$. From $\bullet \notin t$ we infer $t \in T\{t/\bullet\}$ and $(\mathcal{T}\{t/\bullet\})_{\to t} = \mathcal{T}_{\to_t}\{t/\bullet\}$. Hence $\ell \leftrightarrow^*_{\mathcal{T}_{\to t}\{t/\bullet\}} r$ by Lemma 4(2) and thus $\ell' \leftrightarrow^*_{\mathcal{R}'} \ell \leftrightarrow^*_{\mathcal{R}'} r \leftrightarrow^*_{\mathcal{R}'} r'$ as desired. □

The following lifting lemma is the key to the completeness of our enumeration procedure.

Lemma 6. *If $\mathcal{R} = \mathcal{S} \uplus \mathcal{T}_{\to t} \Longrightarrow^* \mathcal{R}'$ with $\bullet_t \notin \mathcal{R}$ and $P(\mathcal{S}\langle\bullet_t\rangle, \{t\} \cup T_{-t}\langle\bullet_t\rangle, \bullet_t)$ then there exist a* TRS *$\mathcal{U}$ and a term v such that $\mathcal{S}\langle\bullet_t\rangle \Longrightarrow^* \mathcal{U}$, $\mathcal{R}' = (\mathcal{U} \cup \mathcal{V}_{\to v})\{v/\bullet_t\}$, $P(\mathcal{U}, V, \bullet_t)$ and $\bullet_t \notin v \in V$ with $V = (\{t\} \cup T_{-t}\langle\bullet_t\rangle)\downarrow_{\mathcal{U}}$.*

Proof. We consider a single step $\mathcal{R} = \mathcal{S} \uplus \mathcal{T}_{\to t} \Longrightarrow \mathcal{R}'$. The statement of the lemma follows by a straightforward induction argument. Let $\ell \to r$ be the rule used in this step. So $\mathcal{R}' = (\mathcal{R} \setminus \{\ell \to r\})\{\ell/r\} \cup \{r \to \ell\}$. We distinguish two cases.

First suppose $\ell \to r \in \mathcal{S}$. So $\mathcal{R}' = (\mathcal{S}\setminus\{\ell \to r\})\{\ell/r\} \cup \mathcal{T}_{\to t}\{\ell/r\} \cup \{r \to \ell\}$. Let $\ell' = \ell\{\bullet_t/t\}$ and $r' = r\{\bullet_t/t\}$. We have $\ell' \to r' \in \mathcal{S}\langle\bullet_t\rangle$ and $r' \not\trianglelefteq \ell'$ follows from $r \not\trianglelefteq \ell$. Hence $\mathcal{S}\langle\bullet_t\rangle \Longrightarrow \mathcal{U}$ for $\mathcal{U} = (\mathcal{S}\langle\bullet_t\rangle \setminus \{\ell' \to r'\})\{\ell'/r'\} \cup \{r' \to \ell'\}$. Define $v = t\{\ell/r\}$. We show $v = t\{\ell'/r'\}$. We distinguish two cases.

- If $r' \trianglelefteq t$ then we must have $\bullet_t \notin r'$ since $\bullet_t \notin t$. Hence $r' = r$. If $\bullet_t \in \ell$ then $\ell = \ell'\{t/\bullet_t\} \trianglerighteq t \trianglerighteq r' = r$, contradicting $r \not\trianglelefteq \ell$. So $\bullet_t \notin \ell$ and thus $\ell' = \ell$.
- If $r' \not\trianglelefteq t$ then $t\{\ell'/r'\} = t$. Suppose $r \trianglelefteq t$. Because the rewrite rule $\ell \to r$ does not belong to $\mathcal{T}_{\to t}$, $r \neq t$ and thus $r \triangleleft t$. Hence $r' = r\langle\bullet_t\rangle = r$, contradicting $r' \not\trianglelefteq t$. So $r \not\trianglelefteq t$ and thus $v = t$.

Using similar reasoning we easily obtain $r'\{t/\bullet_t\} = r'\{v/\bullet_t\}$ and $\ell'\{t/\bullet_t\} = \ell'\{v/\bullet_t\}$. Hence $\ell = \ell'\{v/\bullet_t\}$ and $r = r'\{v/\bullet_t\}$. If we show

$$(\mathcal{S}\langle\bullet_t\rangle \setminus \{\ell' \to r'\})\{\ell'/r'\}\{v/\bullet_t\} = (\mathcal{S} \setminus \{\ell \to r\})\{\ell/r\} \tag{1}$$

and

$$\mathcal{V}_{\to v}\{v/\bullet_t\} = \mathcal{T}_{\to t}\{\ell/r\} \tag{2}$$

then $\mathcal{R}' = (\mathcal{U} \cup \mathcal{V}_{\to v})\{v/\bullet_t\}$ follows. We have $T \subseteq \mathsf{NF}(\mathcal{S})$ by the canonicity of $\mathcal{R}$. Hence also $T\langle\bullet_t\rangle \subseteq \mathsf{NF}(\mathcal{S}\langle\bullet_t\rangle)$. Moreover, $T\langle\bullet_t\rangle \subseteq \mathsf{NF}(\mathcal{S}\langle\bullet_t\rangle\{\ell'/r'\})$ because if $u \in T\langle\bullet_t\rangle \setminus \mathsf{NF}(\mathcal{S}\langle\bullet_t\rangle\{\ell'/r'\})$ then $\ell' \trianglelefteq u$ and hence $\ell = \ell'\langle\bullet_t\rangle \trianglelefteq u\langle\bullet_t\rangle \in T$, contradicting the canonicity of $\mathcal{R}$. As a consequence, $T_{-t}\langle\bullet_t\rangle\downarrow_{\mathcal{U}} = T_{-t}\langle\bullet_t\rangle\{\ell'/r'\}$. So both (1) and (2) follow from the identity

$$u\langle\bullet_t\rangle\{\ell'/r'\}\{v/\bullet_t\} = u\{\ell/r\}$$

for an arbitrary term $u \in \mathsf{LHS}(\mathcal{S} \setminus \{\ell \to r\}) \cup \mathsf{RHS}(\mathcal{S} \setminus \{\ell \to r\}) \cup T_{-t}$ and the observation that the rewrite rules in $\mathcal{V}_{\to v}\{v/\bullet_t\}$ and $\mathcal{T}_{\to t}\{\ell/r\}$ have the same right-hand side $v = t\{\ell/r\}$. If $r' \trianglelefteq u\langle\bullet_t\rangle\{\ell'/r'\}$ then $r \trianglelefteq u$ and $\ell'\{v/\bullet_t\} = \ell$. If an occurrence of $\bullet_t$ in $u\langle\bullet_t\rangle$ is no subterm of r' then it will be replaced by v when computing $u\langle\bullet_t\rangle\{\ell'/r'\}\{v/\bullet_t\}$. In this case the corresponding occurrence of t in u is no subterm of r and so it will be replaced by $t\{\ell/r\} = v$ in the computation of $u\{\ell/r\}$. Finally, Lemma 3 yields $P(\mathcal{U}, V, \bullet_t)$.

Next suppose $\ell \to r \in \mathcal{T}_{\to t}$. So $\ell \in T_{-t}$ and $r = t$. In this case we let $\mathcal{U} = \mathcal{S}\langle\bullet_t\rangle$ and $v = \ell$. We obtain $\bullet_t \notin v$ from $\bullet_t \notin \mathcal{R}$. We have $\mathcal{R}' = \mathcal{S}\{\ell/r\} \cup (\mathcal{T}_{\to t} \setminus \{\ell \to r\})\{\ell/r\} \cup \{r \to \ell\}$. Moreover, $\mathcal{S}\{\ell/r\} = \mathcal{S}\{\ell/t\} = \mathcal{S}\{\bullet_t/t\}\{\ell/\bullet_t\} = \mathcal{U}\{v/\bullet_t\}$. Let $T' = T \setminus \{\ell, r\}$. For $W = \{\ell, r\} \cup T'\{\ell/r\}$, we have $(\mathcal{T}_{\to t} \setminus \{\ell \to r\})\{\ell/r\} \cup \{r \to \ell\} = \mathcal{W}_{\to v}$. Condition 2 of $P(\mathcal{S}\langle\bullet_t\rangle, \{t\} \cup T_{-t}\langle\bullet_t\rangle, \bullet_t)$ yields $\{t\} \cup T_{-t}\langle\bullet_t\rangle \subseteq \mathsf{NF}(\mathcal{S}\langle\bullet_t\rangle) = \mathsf{NF}(\mathcal{U})$ and thus $V = \{t\} \cup T_{-t}\langle\bullet_t\rangle$. We obtain $\mathcal{V}_{\to v}\{v/\bullet_t\} = (\{r\} \cup T_{-r}\{\ell/r\})_{\to v}$. Since $r \not\trianglelefteq \ell$, $\ell \in T_{-r}\{\ell/r\}$. This concludes the proof of $\mathcal{R}' = (\mathcal{U} \cup \mathcal{V}_{\to v})\{v/\bullet_t\}$. We trivially have $P(\mathcal{U}, V, \bullet_t)$. □

Theorem 2. *If $\mathcal{R}$ is a ground canonical* TRS *then* $\mathsf{CaP}(\mathcal{R}) = E(\mathcal{R})$.

Proof. We first prove the soundness of the enumeration procedure, $E(\mathcal{R}) \subseteq \mathsf{CaP}(\mathcal{R})$, by induction on $\#\mathcal{R}$. The inclusion is trivial when $\mathcal{R} = \varnothing$. Let $\#\mathcal{R} > 0$ and $\mathcal{R}' \in E(\mathcal{R})$. So $\mathcal{R}' = (\mathcal{U} \cup \mathcal{V}_{\to v})\{v/\bullet_t\}$ with $t \in \mathsf{RHS}(\mathcal{R})$ and $\bullet_t \notin v \in V = (\{t\} \cup T_{-t}\langle\bullet_t\rangle)\downarrow_{\mathcal{U}}$ where $T = \{t\} \cup \{s \mid s \to t \in \mathcal{R}\}$, $\mathcal{S} = \mathcal{R} \setminus \mathcal{T}_{\to t}$ and $\mathcal{U} \in E(\mathcal{S}\langle\bullet_t\rangle)$. Since $\#\mathcal{S}\langle\bullet_t\rangle = \#\mathcal{S} < \#\mathcal{R}$, we can apply the induction hypothesis to $\mathcal{S}\langle\bullet_t\rangle$. This yields $E(\mathcal{S}\langle\bullet_t\rangle) \subseteq \mathsf{CaP}(\mathcal{S}\langle\bullet_t\rangle)$. So $\mathcal{U}$ is a canonical presentation of $\mathcal{S}\langle\bullet_t\rangle$. Hence $\mathcal{S}\langle\bullet_t\rangle \Longrightarrow^* \mathcal{U}$ by the completeness of $(\star)$. Let $V' = \{t\} \cup T_{-t}\langle\bullet_t\rangle$. We show $P(\mathcal{S}\langle\bullet_t\rangle, V', \bullet_t)$:

1. If $\mathcal{S}\langle\bullet_t\rangle$ is not canonical then $\mathcal{S}$ is not canonical due to Lemma 4(3), contradicting Lemma 4(1).
2. Let $v' \in V'$. If $v' = t$ then $v' \in \mathsf{NF}(\mathcal{S}\langle\bullet_t\rangle)$ because otherwise $t \notin \mathsf{NF}(\mathcal{S})$, contradicting the canonicity of $\mathcal{R}$. If $v' \in T_{-t}\langle\bullet_t\rangle$ then $v'\{t/\bullet_t\}$ is the left-hand side of a rule in $\mathcal{T}_{\to t}$ and hence a normal form of $\mathcal{S}$ by the canonicity of $\mathcal{R}$. Hence $v' = v'\{t/\bullet_t\}\{\bullet_t/t\} \in \mathsf{NF}(\mathcal{S}\langle\bullet_t\rangle)$.
3. Let $v' \in V'$. If $v' = t$ then v' is no subterm of a term in $\mathcal{S}\langle\bullet_t\rangle$ and the same holds for the terms in $T_{-t}\langle\bullet_t\rangle$. Let $v' \in T_{-t}\langle\bullet_t\rangle$. If v' is a subterm of a term in $\mathsf{LHS}(\mathcal{S}\langle\bullet_t\rangle) \cup \mathsf{RHS}(\mathcal{S}\langle\bullet_t\rangle)$ then $v'\{t/\bullet_t\}$ is a subterm of a term in $\mathsf{LHS}(\mathcal{S}) \cup \mathsf{RHS}(\mathcal{S})$. In the preceding case we observed that $v'\{t/\bullet_t\}$ is the left-hand side of a rule in $\mathcal{T}_{\to t}$. Hence we obtain a contradiction with the canonicity of $\mathcal{R}$. If v' is a subterm of a term in $V'_{-v'}$ then it must be a proper subterm and thus $v'\{t/\bullet_t\}$ is a proper subterm of a term in T_{-t}, contradicting canonicity of $\mathcal{R}$.
4. By construction, $\bullet_t \notin \mathsf{LHS}(\mathcal{S}\langle\bullet_t\rangle) \cup \mathsf{RHS}(\mathcal{S}\langle\bullet_t\rangle)$. From $t \notin T_{-t}$ we infer $\bullet_t \notin V' = \{t\} \cup T_{-t}\langle\bullet_t\rangle$.

Lemma 3 now yields $P(\mathcal{U}, V, \bullet_t)$. Hence $\mathcal{R}' = (\mathcal{U} \cup \mathcal{V}_{\to v})\{v/\bullet_t\}$ is canonical by Lemma 2. For $W = \{t\} \cup T_{-t}\langle\bullet_t\rangle$ we can write $\mathcal{R} = \mathcal{S} \uplus \mathcal{T}_{\to t} = (\mathcal{S}\langle\bullet_t\rangle \cup \mathcal{W}_{\to t})\{t/\bullet_t\}$, so that we have $V = W\downarrow_{\mathcal{U}}$. The equivalence of $\mathcal{R}$ and $\mathcal{R}'$ is then an immediate consequence of Lemma 5.

Next we turn to the completeness of the enumeration procedure, $\mathsf{CaP}(\mathcal{R}) \subseteq E(\mathcal{R})$. Again we use induction on $\#\mathcal{R}$. The inclusion is trivial when $\mathcal{R} = \varnothing$. Let $\#\mathcal{R} > 0$ and $\mathcal{R}' \in \mathsf{CaP}(\mathcal{R})$. We have $\mathcal{R} \Longrightarrow^* \mathcal{R}'$ by the completeness of $(\star)$. Let $t \in \mathsf{RHS}(\mathcal{R})$ and decompose $\mathcal{R}$ into $\mathcal{S} \uplus \mathcal{T}_{\to t}$ as usual. Let $\bullet_t$ be a fresh constant. Lemma 6 yields a TRS $\mathcal{U}$ and a term v such that $\mathcal{S}\langle\bullet_t\rangle \Longrightarrow^* \mathcal{U}$, $\mathcal{R}' = (\mathcal{U} \cup \mathcal{V}_{\to v})\{v/\bullet_t\}$ and $\bullet_t \notin v \in V$ with $V = (\{t\} \cup T_{-t}\{t/\bullet_t\})\downarrow_{\mathcal{U}}$. We have $\mathcal{U} \in \mathsf{CaP}(\mathcal{S}\langle\bullet_t\rangle)$ by the soundness of $(\star)$. Since $\#\mathcal{S}\langle\bullet_t\rangle < \#\mathcal{R}$, we can apply the induction hypothesis. This yields $\mathcal{U} \in E(\mathcal{S}\langle\bullet_t\rangle)$. Hence $\mathcal{R}' \in E(\mathcal{R})$ by definition. □

Corollary 1. *If $\mathcal{R}$ is a ground canonical* TRS *then* $\#\,\mathsf{CaP}(\mathcal{R}) \leqslant \mathsf{bound}(\mathcal{R})$.

Proof. We prove $\#\,E(\mathcal{R}) \leqslant \mathsf{bound}(\mathcal{R})$ by induction on $\#\mathcal{R}$. Since $\#\,E(\mathcal{R}) = \#\,\mathsf{CaP}(\mathcal{R})$ according to Theorem 2, this establishes the claim. If $\mathcal{R} = \varnothing$ then $\#\,E(\mathcal{R}) = \#\{\varnothing\} = 1 = \mathsf{bound}(\mathcal{R})$. Suppose $\mathcal{R} \neq \varnothing$. Let $t \in \mathsf{RHS}(\mathcal{R})$ and write $\mathcal{R} = \mathcal{S} \uplus T_t$ where $T = \{t\} \cup \{\ell \mid \ell \to t \in \mathcal{R}\}$. From the construction of $E(\mathcal{R})$ we obtain $\#\,E(\mathcal{R}) \leqslant \#\,T \times \#\,E(\mathcal{S})$. Since $\#\,E(\mathcal{S}) \leqslant \mathsf{bound}(\mathcal{S})$ by the induction hypothesis, $\#\,E(\mathcal{R}) \leqslant \#\,T \times \mathsf{bound}(\mathcal{S})$. Moreover, we have $\mathsf{bound}(\mathcal{R}) = (1+\#\,\mathcal{T}_{\to t}) \times \mathsf{bound}(\mathcal{S})$ from the definition of bound. Together with the obvious $\#\,T = 1+\#\,\mathcal{T}_{\to t}$ we obtain $\#\,E(\mathcal{R}) \leqslant \mathsf{bound}(\mathcal{R})$. □

Example 3. For any of the canonical TRSs $\mathcal{R}$ in Example 1 we have $\mathsf{bound}(\mathcal{R}) = 3 \times 2 = 6$. This is less than the bound 8 from [3] but double the actual size of $\mathsf{CaP}(\mathcal{R})$. The reason for the discrepancy is the existence of non-reversible rules.

References

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

[2] Aart Middeldorp, Masahiko Sakai, and Sarah Winkler. Ground canonical rewrite systems revisited. In *Proceedings of the 12th International Workshop on Confluence*, pages 44–48, 2023. Available at http://cl-informatik.uibk.ac.at/iwc/iwc2023.pdf.

[3] Wayne Snyder. A fast algorithm for generating reduced ground rewriting systems from a set of ground equations. *Journal of Symbolic Computation*, 15(4):415–450, 1993. doi:10.1006/jsco.1993.1029.

On Proving Confluence of Generalized Term Rewriting Systems Using CONFident*

Raúl Gutiérrez and Salvador Lucas

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

Abstract

Generalized Term Rewriting Systems (GTRSs) extend Term Rewriting Systems by providing a highly expressive framework that integrates conditional rules, context-sensitive replacement restrictions, and Horn clauses directly into the rewriting formalism. In this work, we extend our confluence tool CONFident to prove and disprove confluence of GTRSs.

1 Introduction

A *Generalized Term Rewriting System (GTRS [14])* is a tuple $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$, where $\mathcal{F}$ is a signature of *function symbols*; Π is a signature of *predicate symbols*, including $\to$ and $\to^*$; μ is a *replacement map* i.e., for all k-ary function symbols $f \in \mathcal{F}$, $\mu(f) \subseteq \{1, \ldots, k\}$ is the set of *active* arguments on which rewriting steps are allowed [12]; H is a set of definite Horn clauses $A \Leftarrow c$, where the predicate symbol of A is not $\to$ or $\to^*$; and R is a set of rewrite rules $\ell \to r \Leftarrow c$ [14, Definition 51]. In both cases, c is a sequence of atoms.

Example 1. *The GTRS displayed in Figure 1 permits the elimination of duplicate occurrences of numbers* $\mathsf{s}^n(\mathsf{0})$ *in Peano notation from a list [15, Figure 1].*

Term Rewriting Systems (TRSs [1]), Conditional TRSs [9], Context-Sensitive TRSs (CS-TRSs [12]), and Context-Sensitive CTRSs [13, Section 8.1] are particular cases of GTRSs. Furthermore, Horn clauses allow us to model complex characteristics of algebraic programming languages, such as sort information and membership predicates from Maude. Programs of reduction-based systems like Maude [2] can often be encoded as GTRSs [16]. Confluence of GTRSs has been investigated in [14, 15] and the results apply to all aforementioned variants of rewrite systems.

Example 2. *The GTRS* $\mathcal{R}$ *in Figure 1 is shown confluent in [15, Example 9], as a particular case of* weakly V-orthogonal GTRS *[15, Definition 12], which are proved confluent in [15].*

Since CONFident[1] does not support fully general GTRSs yet, this paper explains how it has been extended to handle them.

2 Processors extended in CONFident

CONFident implements the *Confluence Framework* for GTRSs introduced in [6]. Roughly speaking, given a *confluence problem* $CR(\mathcal{R})$ for a GTRS $\mathcal{R}$, the framework applies a number of

*Partially supported by MCIN/AEI project PID2024-162030OB-100 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe" and by the grant CIPROM/2022/6 funded by Generalitat Valenciana.

[1] http://zenon.dsic.upv.es/confident/

```
(REPLACEMENT-MAP
  (from )
  (rm 2)
  (cons 2)
)
(VAR m n x xs y)
(HORN-CLAUSES
  Neq(0,s(n))
  Neq(s(n),0)
  Neq(s(m),s(n))
    | Neq(m,n)
)
```

```
(RULES
  from(n) -> cons(n,from(s(n)))
  rm(x,nil) -> nil
  rm(x,cons(x,xs)) -> rm(x,xs)
  rm(x,cons(y,xs)) -> cons(y,rm(x,xs)) | Neq(x,y)
  edups(nil) -> nil
  edups(cons(x,nil)) -> cons(x,nil)
  edups(cons(x,cons(x,xs))) -> edups(cons(x,xs))
  edups(cons(x,cons(y,xs)))
     -> cons(x,rm(x,edups(cons(y,xs)))) | Neq(x,y)
)
```

Figure 1: The GTRS $\mathcal{R}$ in [15, Figure 1] in the extended COPS format in [4]

processors which transform the problem into auxiliary problems where "simpler" properties of $\mathcal{R}$ like *local confluence* $WCR(\mathcal{R})$, *strong confluence* $SCR(\mathcal{R})$, or *joinability* properties of the *conditional pairs* $\pi : \langle s, t\rangle \Leftarrow c$ for terms s and t and (possibly empty) conditional part c (e.g., Critical Pairs [8], Conditional Critical Pairs [10], Conditional Variable Pairs, or just CVPs, [14], Parallel CVPs [15], etc.) which are used to prove the aforementioned confluence properties of $\mathcal{R}$ (e.g., *joinability* $JO(\mathcal{R}, \pi)$ or *strong joinability* $SJO(\mathcal{R}, \pi)$) are attempted.

Although [6] demonstrates the treatment of confluence proofs for TRSs, CS-TRSs, CTRSs, and CS-CTRSs only, the underlying processors of the framework are readily generalizable to fully general GTRSs. For instance, the well-known Knuth-Bendix result [11, Theorem 4] showing that terminating TRSs that satisfy a *lattice condition* (i.e., Huet's local confluence [8]) and the characterization of local confluence of TRSs as the joinability of critical pairs [8, Lemma 3.1] has been extended and generalized to CS-TRSs, CTRSs and GTRSs in [18, 14] (by considering not only CPs but also other *conditional pairs*). Thus, we use a single processor P_{HE} to apply the results by different authors which permit a proof of local or strong confluence on the basis of checking (strong) joinability of conditional pairs [6, Section 5.3]. Actually, such joinability checkings are forwarded to processor P_{JO} [6, Section 5.5], which is in charge of checking the corresponding joinability property depending on the considered joinability problem $JO(\mathcal{R}, \pi)$ or $SJO(\mathcal{R}, \pi)$.

The integration of the new weak V-orthogonality results in [15] is made through the already existing processor P_{Orth} implementing the usual confluence results for orthogonal systems [6, Section 5.4.3]. We only needed to include an additional option in the definition of the processor to treat weak V-orthogonality according to [15, Definition 12].

3 Implementation

CONFident is implemented in Haskell. The extension that adds Horn clauses in CONFident needed changes in 37 modules and around 600 *loc*. The parser has been extended to accept the COPS notation[2] enriched with a new block `HORN-CLAUSES` to specify them (see Figure 1). These new predicates can be used both in the conditions of the Horn clauses and in the rules. To implement the framework, the following data type is used:

```
1 -- | Parameterized generalized term rewriting system.
```

[2] https://project-coco.uibk.ac.at/problems/

```
data GTRS a b c
  = GTRS { ctrsSignature     :: Signature a
         , ctrsVariables     :: Variables b
         , ctrsRules         :: Set (CRule c)
         , ctrsHornClauses   :: Set (HornClause c)
         }
```

Function symbols and variables are incrementally composed in accordance with the required characteristics. This construction mechanism is employed throughout all of our tools. In this way, a function symbol have the type `IdFCS IdFTRS` and the following structure:

```
-- | Term rewriting system function symbol
data IdFTRS
  = IdFTRS { fid       :: Int          -- ^ Identifier
           , fname     :: String       -- ^ Name of the function symbol
           , arity     :: Int          -- ^ Arity of the symbol
           }
-- | Context-Sensitive function symbol
data IdFCS id
  = IdFCS { csfprev    :: id           -- ^ Previous info
          , mu         :: [Int]        -- ^ Active positions
          }
```

This approach enables the definition of extended function symbols and variables enriched with additional characteristics. This "Russian doll" style construction facilitates the simple introduction of new extensions into the system, such as axioms or sort information. Nevertheless, the computational treatment remains challenging; for example, moving from syntactic unification to order-sorted unification may have important consequences in the analysis of critical pairs, for instance.

This approach allows us to define extended function symbols (and variables) enriched with additional characteristics, such as axioms or sort information. Using this russian doll definition, extensions are simple to be introduced in the system. Of course, the computational treatment is different (for example, it is not trivial to pass from the syntactic unification to teh order-sorted unification). For variables, `IdVTRS` is sufficient for GTRSs:

```
-- | Term rewriting system variable symbol
data IdVTRS
  = IdVTRS { vid       :: Int          -- ^ Identifier
           , vname     :: Maybe String -- ^ Name of the variable
           }
```

Accordingly, our specific GTRS is defined as follows:

```
-- | 'InGTRS' are the GTRS we use in our problems
type InGTRS = GTRS (IdFCS IdFTRS) IdVTRS (Term (IdFCS IdFTRS) IdVTRS)
```

Conditional rules and Horn clauses are constructed over terms. To support predicates, the set of possible conditions has been extended as follows:

```
-- | Basic rule
data Rule a = a :-> a
-- | Basic atom
data Atom a = Pred a
-- | A condition
data Cond a = a :->* a     -- rewriting in zero or more steps
```

```
            ...              -- more rewriting relations
          | PredCond a       -- predicate condition
-- | Conditional rule
data CRule a = Rule a :=> [Cond a]
-- | Horn Clause
data HornClause a = Atom a :<= [Cond a]
```

Consequently, conditional rules and Horn clauses are represented by the types `CRule (Term (IdFCS IdFTRS) IdVTRS)` and `HornClause (Term (IdFCS IdFTRS) IdVTRS)`, respectively.

We use the `muterm-framework` library to define instances of problems, processors, and also strategy combinators to easily define our strategy. In the confluence framework [7], we consider two kinds of problems: *confluence problems* and *joinability problems*. Our new GTRS representation is integrated into the framework by providing the appropriate instances of these type classes. `InCritPair` represents the information associated with a computed critical pair.

```
-- | Confluence Problem Type Class
class IsConfProblem typ trs | typ -> trs where
  getConfProblem :: typ -> trs
  setConfProblem :: trs -> typ -> typ
-- | Conditional Joinability Problems have a list of critical pairs
class IsJProblem typ cps | typ -> cps where
  getCPairs :: typ -> [cps]
  setCPairs :: [cps] -> typ -> typ
-- | Confluence Problem - GTRS
data InConfGTRS gtrs = InConfGTRS gtrs
-- | Joinability Problem - GTRS
data InJGTRS trs cps = InJGTRS trs [cps]
-- | GTRSs are Confluence Problems
instance IsConfProblem (InConfGTRS InGTRS) InGTRS where
  getConfProblem (InConfGTRS trs) = trs
  setConfProblem trs (InConfGTRS _) = InConfGTRS trs
-- | GTRSs Problems together with CPs are Joinability Problems
instance IsJProblem (InJGTRS InGTRS InCritPair) InCritPair where
  getCPairs (InJGTRS _ cps) = cps
  setCPairs cps (InJGTRS gtrs _) = InJGTRS gtrs cps
```

After computing the critical pairs, confluence problems are transformed into joinability problems.

Problem categories are encoded as data types. The associated data type determines the strategy that is applied to a problem.

```
-- | GRewriting problem
data GRewriting = GRewriting
-- | JGRewriting problem
data JGRewriting = JGRewriting
class Functor (Problem typ) => IsProblem typ where
  data Problem typ :: * -> *
  getProblemType :: Problem typ trs -> typ
  getR       :: Problem typ trs -> trs
-- | GRewriting problem is a Problem
instance IsProblem GRewriting where
  data Problem GRewriting a = GRew a
  getProblemType (GRew _) = CRewriting
  getR (GRew r) = r
```

```
-- | JGRewriting problem is a Problem
instance IsProblem JGRewriting where
  data Problem JGRewriting a = JGRew a
  getProblemType (JGRew _) = JGRewriting
  getR (JGRew r) = r
```

Confluence and joinability problems are instances of a general problem abstraction (`IsProblem`), defined by the presence of an associated term rewriting system, and are handled by corresponding specialized *processors*. Processors can be *pure* if they transform a problem into a (hopefully simpler) problem of the same type or *transformational*, if the type of the problem changes. We show an example of both.

```
-- Infeasibility Rule Processor
data InfGTRSProcessor = InfGTRSProcessor {timeout :: Maybe Int}
-- Processor
instance (... functional dependencies ...)
  => Processor InfGTRSProcessor (Problem GRewriting conftrs)
                                (Problem GRewriting conftrs) where
  apply InfGTRSProcessor {timeout = to} inP = ...
-- | Critical Pairs Processor
data CriticalPairGTRSProcessor = CriticalPairGTRSProcessor
-- Processor
instance (... functional dependencies ...)
  => Processor CriticalPairGTRSProcessor (Problem GRewriting conftrs1)
                                         (Problem JGRewriting conftrs2) where
  apply CriticalPairsCProcessor inP = ...
```

Each processor has its own name. The strategy combines the different processors in order to find a proof tree. The strategy is the same as in [6], with infeasibility, critical pairs and termination processors adapted to GTRSs.

By describing Horn clauses independently instead of encoding them as rewriting rules, we can in practice simplify the conditions for critical pairs, avoiding the need to use rewriting when it is not required, and improve the flexibility of processors based on polynomial interpretations, as predicates may be interpreted independently, as shown in [17].

4 Benchmarks

We have performed our benchmarks using 24 GTRS examples from the literature.[3] We are able to prove confluence of 15 of them and non-confluence of 4. Overall, we are able to deal with almost 80% of the considered examples. These results are promising, as this first (full) extension of CONFident to GTRSs essentially relies on improvements recently introduced in our tools (infChecker [5] and MU-TERM GTRS [4]) rather than on techniques specific to GTRSs.

5 Conclusions and Future Work

We have extended CONFident to give support to fully general GTRSs in proofs of confluence. The tool implements the Confluence Framework for proving confluence of GTRSs [6], allowing the use of infeasibility processors [3] to deal with proofs of joinability of conditional pairs involving atomic components in conditions, and termination processors that take Horn clauses into

[3] http://zenon.dsic.upv.es/confident/benchmarks.html

account, like the ones implemented in MU-TERM GTRS [4]. The benchmarks display promising results.

As for future work, we plan to extend more sophisticated processors to deal with GTRSs which can be used as the basis of a tool for proving confluence of Maude programs.

Acknoledgements. We thank the reviewers for their remarks and comments leading to several improvements.

References

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

[2] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[3] Raúl Gutiérrez and Salvador Lucas. Automatically Proving and Disproving Feasibility Conditions. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 416–435. Springer, 2020.

[4] Raúl Gutiérrez and Salvador Lucas. MU-TERM GTRS: A tool for proving termination of Generalized Term Rewriting Systems. In Dieter Hofbauer and Johannes Waldmann, editors, *20th International Workshop on Termination, WST 2025*, pages 73–78, 2025.

[5] Raúl Gutiérrez and Salvador Lucas. Proving and disproving feasibility with infChecker. In Raúl Gutérrez and Naoki Nishida, editors, *14th International Workshop on Confluence, IWC 2025*, pages 38–44, 2025.

[6] Raúl Gutiérrez, Salvador Lucas, and Miguel Vítores. Proving Confluence in the Confluence Framework with CONFident. *Fundam. Informaticae*, 192(2):167–217, 2024.

[7] Raúl Gutiérrez, Miguel Vítores, and Salvador Lucas. Confluence Framework: Proving Confluence with CONFident. In Alicia Villanueva, editor, *Logic-Based Program Synthesis and Transformation - 32nd International Symposium, LOPSTR 2022, Proceedings*, volume 13474 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2022.

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

[9] Stéphane Kaplan. Conditional Rewrite Rules. *Theor. Comput. Sci.*, 33:175–193, 1984.

[10] Stéphane Kaplan. Simplifying Conditional Term Rewriting Systems: Unification, Termination and Confluence. *J. Symb. Comput.*, 4(3):295–334, 1987.

[11] Donald E. Knuth and Peter E. Bendix. Simple Word Problems in Universal Algebra. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

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

[13] Salvador Lucas. Applications and extensions of context-sensitive rewriting. *Journal of Logical and Algebraic Methods in Programming*, 121:100680, 2021.

[14] Salvador Lucas. Local confluence of conditional and generalized term rewriting systems. *Journal of Logical and Algebraic Methods in Programming*, 136:paper 100926, pages 1–23, 2024.

[15] Salvador Lucas. Confluence of Almost Parallel-Closed Generalized Term Rewriting Systems. In Clark Barrett and Uwe Waldmann, editors, *Automated Deduction - CADE 30 - 30th International Conference on Automated Deduction, Proceedings*, volume 15493 of *Lecture Notes in Artificial Intelligence*, pages 187–206. Springer Nature Switzerland, 2025.

[16] Salvador Lucas. Maude specifications as equational generalized term rewriting systems: Functionalmodules. In Fernando Sáenz-Pérez, editor, *Proceedings XXV Jornadas sobre Programación y Lenguajes, PROLE 2026, Alicante, Spain, 16-18th July 2026*, pages 1–17. Sistedes, 2026. handle: 11705/PROLE/2026/10.

[17] Salvador Lucas and Raúl Gutiérrez. Automatic Synthesis of Logical Models for Order-Sorted First-Order Theories. *J. Autom. Reason.*, 60(4):465–501, 2018.

[18] 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.

Normalised Completion for Stratified Linear Rewriting Systems

Zuan Liu and Philippe Malbos

Université Claude Bernard Lyon 1, CNRS, Institut Camille Jordan, F-69622 Villeurbanne, France
zliu@math.univ-lyon1.fr, malbos@math.univ-lyon1.fr

Abstract

We present a stratified normalisation completion procedure for rewriting systems over linear precategories. Our approach relies on a stratification of the set of rewriting rules according to their confluence and termination properties. We introduce stratified termination functions to establish termination for such systems. We illustrate the method on several classes of algebraic structures, including associative and diagrammatic algebras. In these contexts, we show how the stratification of defining rules can be used to compute hom-bases effectively.

1 Introduction

This work is part of a broader program on rewriting methods in algebra. Rewriting theory provides effective tools to address fundamental problems such as the word problem, the computation of linear bases [24] and resolutions [2, 22, 11] of algebraic structures, and the proof of coherence results [23, 8]. These methods apply to a wide range of algebraic structures, including monoids, algebras over operads, and higher-dimensional structures presented by linear operads or monoidal categories [1, 6, 20, 9]. Most approaches rely on transforming systems of algebraic equations into confluent rewriting systems via completion procedures.

In practice, one often needs to work modulo a set of axioms, either because no terminating orientation exists or to perform constructions modulo these axioms (e.g., for computing linear bases). Rewriting modulo has been extensively studied, both in rewriting theory [12, 19] and in the theory of Gröbner–Shirshov bases [5, 21]. Several completion procedures extend the principle of Knuth–Bendix completion [15] to rewriting modulo axioms [13, 18, 14]. A key insight from these approaches is that, in some situations, it is useful to stratify rewriting rules and compute completion layer by layer, taking into account the normalisation of the lower strata. This allows the overall problem to be decomposed into a series of local steps [18, 14].

We present a method for constructing convergent presentations of diagrammatic algebras arising from linear monoidal categories defined by generators and relations. Our goal is to compute linear hom-bases from such convergent presentations [1, 6, 20, 9]. The main difficulty is the complexity of these specifications, which involve many axioms together with intricate interactions between categorical and linear structures. Establishing termination and confluence is particularly challenging due to the two-dimensional combinatorial nature of the rewritten objects, namely 2-cells subject to interchange relations. Our approach is based on decomposing rewriting systems into strata according to termination and confluence properties. We develop techniques adapted to this setting to prove termination and perform completion layer by layer.

The present contribution focuses on the resulting stratified completion procedure. Our constructions are formulated for rewriting systems in linear higher precategories, presented by linear $(n+1)$-prepolygraphs, as recalled in Section 2. This framework encompasses, in particular, associative algebras and diagrammatic algebras. Section 3 introduces a normalised completion procedure based on a stratification of prepolygraphs. It proceeds stratum by stratum: at each

step, context-induced reductions are normalised with respect to lower strata and oriented by a monomial order, yielding the next stratum. This normalisation makes the confluence property modular [4, Sec. 9.3]: if each normalised stratum is confluent, then their union is confluent. Finally, we illustrate the effectiveness of the procedure on algebraic examples.

2 Termination of stratified algebraic rewriting systems

Rewriting in precategories. Our results are formulated in the setting of free linear higher precategories [7], a linear version of precategories introduced in [9, 20]. For $n \geq 1$, a *linear n-precategory* is a linear n-category in which the interchange law is not imposed, that is, the $\circ_{n-1}$-composition of n-cells is not required to be functorial. We recall below the notation needed in the sequel and refer to the references above for the definitions.

In the sequel, $\mathcal{C}$ denotes a free linear n-precategory over a field $\Bbbk$ generated by a linear n-prepolygraph. Rewriting is performed modulo the underlying linear structure, in the spirit of Gröbner–Shirshov bases [10]. A *context* of $\mathcal{C}$ is a pair $(\Box, C)$ consisting of an $(n-1)$-sphere $\Box$ of $\mathcal{C}$ and an n-cell $C[\Box]$ that contains exactly one occurrence of $\Box$. From [7], every monomial n-cell h in $\mathcal{C}$ admits a unique decomposition $h = C_1[\alpha_1] \circ_{n-1} \cdots \circ_{n-1} C_m[\alpha_m]$, where α_i are generating n-cells in $\mathcal{C}$ and m is the *length of* h, denoted by $|h|$. Any n-cell f can be decomposed as a linear combination $f = \sum \lambda_i m_i$ of monomials m_i, with $\lambda_i \in \Bbbk$. We write $\mathrm{supp}(f)$ for the set of monomials occurring in f with nonzero coefficient. A *(cellular) extension* of $\mathcal{C}$ is a set X equipped with a map $\rho : X \to \mathrm{Sph}_n(\mathcal{C})$, where $\mathrm{Sph}_n(\mathcal{C})$ consists of pairs (f, g) of parallel n-cells. The elements of X are called *X-rewriting rules*, and are denoted by $\alpha : s\alpha \to t\alpha$. Without loss of generality, we may assume that the source of any rule is a monomial. We denote $\partial\alpha := s\alpha - t\alpha$. The *monomial extension of* X is $X^{\mathtt{m}} := \{(s\alpha, v) \mid \alpha \in X, v \in \mathrm{supp}(t\alpha)\}$.

An *X-rewriting step* $u \underset{X}{\Rightarrow} v$ is an $(n+1)$-cell $\lambda C[\alpha] + 1_f$ in the free $(n+1)$-precategory $\mathcal{C}[X]$ generated by X on $\mathcal{C}$, where $\lambda \in \Bbbk \setminus \{0\}$, $\alpha \in X$, and $sC[\alpha] \notin \mathrm{supp}(f)$, such that $u = \lambda C[s\alpha] + f$ and $v = \lambda C[t\alpha] + f$. An *$X$-rewriting path* $u \underset{X}{\Rightarrow\!\!\!\!\Rightarrow} v$ is an $(n+1)$-cell in the $(n+1)$-precategory $\mathcal{C}[X]$ that is a composite $\sigma = \sigma_1 \circ_n \ldots \circ_n \sigma_p$ of X-rewriting steps such that $s\sigma_1 = u$ and $t\sigma_p = v$.

An n-cell f in $\mathcal{C}$ is in *X-normal form* if it cannot be reduced by any X-rewriting step. We denote by $\mathrm{Nf}(X)$ the set of X-normal forms in $\mathcal{C}$, and by $\mathrm{Nf}_m(X)$ its subset of monomials. An n-cell g is a *normal form of an n-cell f* if $f \underset{X}{\Rightarrow\!\!\!\!\Rightarrow} g$ and g is in X-normal form. In this case, we write $g = f{\downarrow_X}$. If X is *normalising*, an *X-normalisation strategy* is a map $(-){\downarrow_X}$ sending every n-cell f of $\mathcal{C}$ to a normal form $f{\downarrow_X}$. Any context C admits a unique decomposition $f_n \circ_{n-1} \cdots \circ_1 f_1 \circ_0 \Box \circ_0 g_1 \circ_1 \cdots \circ_{n-1} g_n$, with i-cells f_i and g_i in $\mathcal{C}$. We write $C \in \mathrm{Nf}(X)$ if $f_n, g_n \in \mathrm{Nf}(X)$ and set the length $|C| := |f_n| + |g_n|$. A context C is *X-admissible* with α if $C \in \mathrm{Nf}(X)$ and $C[s\alpha] \notin \mathrm{Nf}(X)$. We denote the set of such contexts by $\mathbf{Cont}(\alpha, X)$. We define the *context relation* $\sqsubseteq$ on $\mathcal{C}$ by $\beta_1 \sqsubseteq \beta_2$ if $\beta_2 = C[\beta_1]$ for some context C. A *hom-set total order* $\preccurlyeq$ on $\mathcal{C}$ is a *monomial order* if it is monotone and well-founded. We write $\mathrm{lm}_{\preccurlyeq}(f)$ for the $\preccurlyeq$-maximal monomial in $\mathrm{supp}(f)$, and $\mathrm{lc}_{\preccurlyeq}(f)$ for its coefficient.

Stratified rewriting systems. A *filtration of length k* of an extension X on $\mathcal{C}$ is a nested sequence of subextensions $F^*X : F^1X \subseteq \ldots \subseteq F^kX = X$. We set $F^1_sX := F^1X$, and for $i \geq 2$, we set $F^i_sX := F^iX \setminus F^{i-1}X$, called the *$i$-th stratum* of F^*X, so that $X = F^1_sX \sqcup F^2_sX \sqcup \ldots \sqcup F^k_sX$. Such a decomposition is called a *stratification of* X, denoted by F^*_sX. An extension X equipped with F^*_sX is called a *k-stratified rewriting system*, k-StRS for short.

A k-StRS F^*_sX is *normalising* if F^i_sX is normalising for every $1 \leq i \leq k$. A *normalisation strategy* for a normalising k-StRS F^*_sX is a family of normalisation strategies $(-){\downarrow_i}$ on F^i_sX, for $1 \leq i \leq k$, denoted by $(-){\downarrow_*}$.

A *termination function* for a k-StRS F_s^*X is a family of maps $(\delta_i : \mathcal{C}_n \to (Z_i, \preccurlyeq_i))_{1\leq i\leq k}$, where each $(Z_i, \preccurlyeq_i)$ is a well-founded total preordered set, satisfying the following conditions:

i) $\delta_i(f) \preccurlyeq_i \delta_i(g)$ implies $\delta_i(C[f]) \preccurlyeq_i \delta_i(C[g])$, for all parallel n-cells f, g and context C in $\mathcal{C}$,

ii) $\delta_i(t\alpha) \prec_i \delta_i(s\alpha)$, for all $\alpha \in (F_s^iX)^{\mathtt{m}}$ and $1 \leq i \leq k$,

iii) $\delta_i(t\beta) \preccurlyeq_i \delta_i(s\beta)$, for all $\beta \in (F^{i-1}X)^{\mathtt{m}}$ and $2 \leq i \leq k$.

The following theorem gives a sufficient condition under which the termination property of an StRS is modular. Its proof is given in [9].

Theorem 1. *If a k-StRS F_s^*X admits a termination function, then it is terminating.*

Example 1. Convergent rewriting systems have been widely used in algebra to compute linear bases of algebras [24]. Stratification provides a natural framework for addressing this problem. For instance, consider the associative algebra, seen as a linear 1-category with a single 0-cell, generated by x, y, z and presented by the rules

$$\gamma_1 : xy \to yx, \qquad \gamma_2 : yz \to zy, \qquad \gamma_3 : xz \to zx, \qquad \alpha : xyz \to x^2 + y^2 + z^2.$$

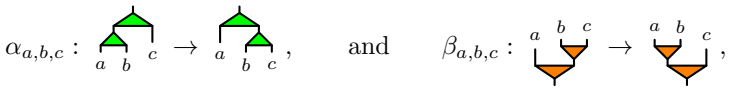
The two systems of rules $\{\gamma_1, \gamma_2, \gamma_3\}$ and $\{\alpha\}$ are individually confluent. To construct a convergent presentation from these rules, we consider the 2-StRS F_s^*X generated by the 1-cells x, y, z, with

$$F_s^1X = \{\gamma_1, \gamma_2, \gamma_3\} \qquad \text{and} \qquad F_s^2X = \{\alpha\}.$$

In the next section, we study the modularity of confluence for this system.

Example 2. We now present an example arising from monoidal categories, formulated in the diagrammatic language of three-dimensional rewriting [3, Chap. 10]. Let $\mathcal{C}$ be the free linear 2-category with a single 0-cell, generated by  and  , for all $a, b \in \mathbb{N}^+$, and presented by the following *interchange relations* $\mathbf{Int}(\mathcal{C})$:

We also impose the *associativity* and *coassociativity* relations



for all $a, b, c > 0$. The two systems of rules $\mathbf{Int}(\mathcal{C})$ and $\{\alpha_{a,b,c}, \beta_{a,b,c}\}$ are individually confluent. We therefore consider the linear 2-precategory presented by the 2-StRS F_s^*X with

$$F_s^1X = \mathbf{Int}(\mathcal{C}) \qquad \text{and} \qquad F_s^2X = \{\alpha_{a,b,c}, \beta_{a,b,c}\}.$$

In the next section, we compute a linear hom-basis for the category $\mathcal{C}$ by considering the union of these two confluent rewriting extensions.

3 Stratified normalisation completion procedure

In this section, we introduce a normalisation completion procedure for an StRS.

Normalisation. A *normalisation* of a k-StRS F_s^*X, with respect to an X-compatible monomial order $\preccurlyeq$, is a k-StRS $F_s^*\widehat{X}$ defined inductively by the following procedure.

We set $F_s^1\widehat{X} \coloneqq F_s^1X$. We assume that, for $1 \leq j \leq i$, $F^j\widehat{X}$ has been defined, and we choose a normalisation strategy $(-)\downarrow_i$ on $F^i\widehat{X}$. We then define the stratum $F_s^{i+1}\widehat{X}$ as follows.

For every $\alpha \in F_s^{i+1}X$ and context C such that $C[\partial\alpha]\downarrow_i \neq 0$, we define the rule $C[\alpha]\downarrow_i$ on $F^i\widehat{X}$-normal forms, by setting $sC[\alpha]\downarrow_i \coloneqq \mathrm{lm}_{\preccurlyeq}(C[\partial\alpha]\downarrow_i)$ and

$$tC[\alpha]\downarrow_i \coloneqq \begin{cases} \lambda^{-1}C[\partial\alpha]\downarrow_i + \omega, & \text{if } \omega \in \mathrm{supp}(C[t\alpha]\downarrow_i) \setminus \mathrm{supp}(C[s\alpha]\downarrow_i), \\ \lambda^{-1}C[-\partial\alpha]\downarrow_i + \omega, & \text{otherwise,} \end{cases}$$

where $\omega = sC[\alpha]\downarrow_i$ and $\lambda = \mathrm{lc}_{\preccurlyeq}(C[\partial\alpha]\downarrow_i)$. We define the following cellular extension on $\mathcal{C}$ consisting of such rules, one for each admissible context of α:

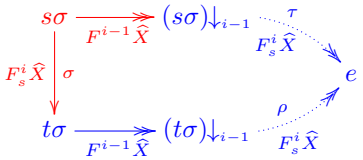
$$[\![\alpha]\!]\downarrow_i \coloneqq \{C[\alpha]\downarrow_i \mid C \in \mathbf{Cont}(\alpha, F^i\widehat{X})\}.$$

The rules in $[\![\alpha]\!]\downarrow_i$ are called the F^iX*-normal reductions* of α. Let $\min_{\sqsubseteq}[\![\alpha]\!]\downarrow_i$ denote the set of minimal rules in $[\![\alpha]\!]\downarrow_i$ with respect to the context relation $\sqsubseteq$. We then define

$$F_s^{i+1}\widehat{X} \coloneqq \bigsqcup_{\alpha \in F_s^{i+1}X} \min_{\sqsubseteq}[\![\alpha]\!]\downarrow_i .$$

We call this procedure the *stratified normalisation procedure*. Given a k-StRS F_s^*X equipped with a well-founded order $\preccurlyeq$, we denote by $\mathtt{SN}(F_s^*X, \preccurlyeq, \downarrow_*)$ the normalisation $F_s^*\widehat{X}$ obtained by applying this procedure to F_s^*X, with respect to a normalisation strategy $(-)\downarrow_*$ on $F_s^*\widehat{X}$.

Lemma 1. *For every $F_s^i\widehat{X}$-rewriting step $\sigma = \mu C[\alpha] + 1_f$, with $i > 1$, if $C[s\alpha]$ is $F^{i-1}\widehat{X}$-reducible and $F_s^i\widehat{X}$ is confluent, there exist two $F_s^i\widehat{X}$-rewriting paths τ and ρ (possibly identities) as in the side diagram.*



For the proof, see [9]. The following lemma shows the modularity of the confluence property of the k-StRS F_s^*X. Its proof is given in [9].

Lemma 2. *Let $1 \leq j \leq k$. If $F_s^i\widehat{X}$ is confluent for every $i \leq j$, then $F^j\widehat{X}$ is confluent.*

In particular, the extension $\widehat{X} = F^k\widehat{X}$ is confluent. Moreover, since any normalisation $F^*\widehat{X}$ is Tietze equivalent to F^*X, this procedure can be used to compute linear bases. The following theorem provides an instance of this approach for associative algebras and linear monoidal categories.

Theorem 2. *Let X be an StRS that presents an associative algebra (resp. a linear monoidal category) $\mathcal{C}$. If there exists an X-compatible monomial order $\preccurlyeq$ and a stratification F_s^*X such that each stratum of the normalisation $F_s^*\widehat{X}$ is confluent, then the set $\mathrm{Nf}_m(\widehat{X})$ forms a linear basis (resp. linear hom-basis) of $\mathcal{C}$.*

The stratified normalisation procedure does not terminate in general, due to the need to explore an infinite set of admissible contexts. For string rewriting, this set can be constructed

recursively, as illustrated below. For diagrammatic rewriting, however, this becomes more difficult. We therefore introduce an effective way to explore this set. We decompose $\min_{\sqsubseteq}[\![\alpha]\!]\downarrow_i$ by the length of admissible contexts and define $F^n \min_{\sqsubseteq}[\![\alpha]\!]\downarrow_i$ as the $\sqsubseteq$-minimal subset of

$$\{C[\alpha]\downarrow_i \mid C \in \mathbf{Cont}(\alpha, F^i\widehat{X}) \text{ and } |C| \leq n\}.$$

We have the following result. For the proof, see [9].

Proposition 1. *Let F_s^*X be an StRS and $\alpha \in F_s^iX$. Assume that $|s\beta| = |t\beta|$ for every $\beta \in F^{i-1}\widehat{X}$. If there exists a natural number n such that*

$$n \geq \max_{\beta \in F^{i-1}\widehat{X}} \{|s\beta| - |\alpha|\} \quad \text{and} \quad F^n \min_{\sqsubseteq}[\![\alpha]\!]\downarrow_{i-1} = F^{n+1} \min_{\sqsubseteq}[\![\alpha]\!]\downarrow_{i-1},$$

then we have $\min_{\sqsubseteq}[\![\alpha]\!]\downarrow_{i-1} = F^n \min_{\sqsubseteq}[\![\alpha]\!]\downarrow_{i-1}$.

Stratified normalisation completion procedure. Combining stratified normalisation with the *Knuth–Bendix completion procedure* yields the following completion procedure for stratified rewriting systems.

Input:
A k-StRS F_s^*X;
An X-compatible monomial order $\preccurlyeq$.
Output: A confluent stratified completion of F_s^*X.
$F_s^*\widehat{X} \leftarrow \mathtt{SN}(F_s^*X, \preccurlyeq, \downarrow_*)$
while $F_s^*\widehat{X}$ *is not confluent* **do**
  $I \leftarrow \{j \in \{1, \dots, k\} \mid F_s^j\widehat{X}$ is not confluent$\}$
  for $j \in I$ **do**
    $F_s^j\widehat{X} \leftarrow \mathtt{KB}(F_s^j\widehat{X}, \preccurlyeq, \downarrow_j)$
  $F_s^*\widehat{X} \leftarrow \mathtt{SN}(F_s^*\widehat{X}, \preccurlyeq, \downarrow_*)$
return $F_s^*\widehat{X}$

By Theorem 2, whenever this procedure terminates, it returns a confluent StRS $F_s^*\widehat{X}$. We now illustrate it through several applications.

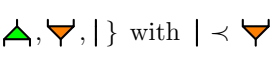
Example 3. Let F_s^*X be the 2-StRS introduced in Example 1, and let $\preccurlyeq$ be the lexicographic order on the alphabet $\{x, y, z\}$ with $z \prec y \prec x$. The normalisation $F_s^*\widehat{X}$ is obtained by setting $F_s^1\widehat{X} := F_s^1X$. As $\mathrm{Nf}(F^1\widehat{X}) = \{z^iy^jx^\ell \mid i, j, \ell \geq 0\}$, every admissible context is of the form $C = z^{i_1}y^{j_1}x^{\ell_1}\square z^{i_2}y^{j_2}x^{\ell_2}$, with $i_1, j_1, \ell_1, i_2, j_2, \ell_2 \geq 0$. By setting $i = i_1 + i_2$, $j = j_1 + j_2$, and $\ell = \ell_1 + \ell_2$, we deduce that

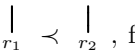
$$[\![\alpha]\!]\downarrow_1 = \{z^{i+1}y^{j+1}x^{\ell+1} \to z^iy^jx^{\ell+2} + z^iy^{j+2}x^\ell + z^{i+2}y^jx^\ell \mid i, j, \ell \geq 0\}$$

and we set

$$F_s^2\widehat{X} := \min_{\sqsubseteq}[\![\alpha]\!]\downarrow_1 = \{\, zy^{j+1}x \to y^jx^2 + y^{j+2} + z^2y^j \mid j \geq 0 \,\}.$$

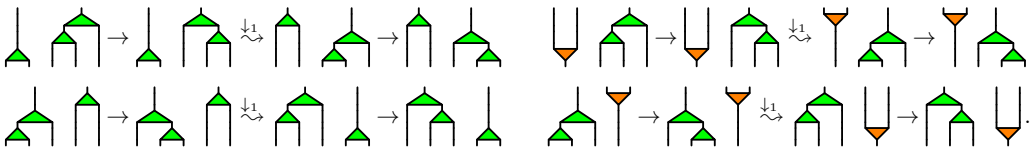
The extension $F_s^1\widehat{X}$ is confluent and $F_s^2\widehat{X}$ has no critical branching, hence is confluent. By Theorem 2, the associative algebra presented by F_s^*X admits the linear basis $\mathrm{Nf}_m(\widehat{X}) = \{\, z^ix^j,\ y^ix^j,\ z^iy^j \mid i, j \geq 0 \,\}$.

Example 4. Consider the 2-StRS F_s^*X introduced in Example 2. Let $\preccurlyeq$ be the lexicographic order on the alphabet $\{$  $\}$ with and  ,  and



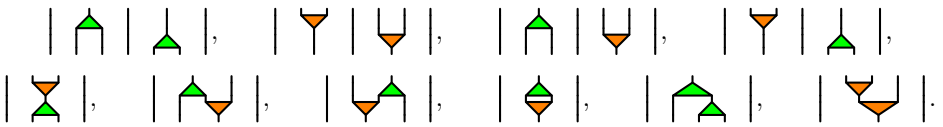
, for all $a+b<c+d$, or $a+b=c+d$ and $b<d$, and $r_1 \prec r_2$. We compute the normalisation with respect to this monomial order, first on 2-cells of length 1, and then extend it to arbitrary 2-cells [9].

We set $F_s^1\widehat{X} = F_s^1 X$. Since $s\alpha \in \mathrm{Nf}(F^1\widehat{X})$, we have $F^0 \min_{\sqsubseteq}[\![\alpha]\!]{\downarrow_1}{=} \{\alpha\}$. For every context C of length 1, $C[s\alpha]$ belongs to $\mathrm{Nf}(F^1\widehat{X})$, except in the following four cases:



In each of these cases, the $\sqsubseteq$-minimal element of the F^1X-normal reduction is α. Thus $F^1 \min_{\sqsubseteq}[\![\alpha]\!]{\downarrow_1}{=} \{\alpha\} = F^0 \min_{\sqsubseteq}[\![\alpha]\!]{\downarrow_1}$. Since $|\alpha| = |\beta| = 2$, we obtain $\min_{\sqsubseteq}[\![\alpha]\!]{\downarrow_1}{=} \{\alpha\}$ by Proposition 1. Similarly, $\min_{\sqsubseteq}[\![\beta]\!]{\downarrow_1}{=} \{\beta\}$. Hence $F_s^2\widehat{X} = F_s^2 X$.

Both $F_s^1\widehat{X}$ and $F_s^2\widehat{X}$ are confluent. By Theorem 2, the set $\mathrm{Nf}_m(\widehat{X})$ forms a hom-basis of $\mathcal{C}$; equivalently, every 2-cell admits a unique decomposition $f_1 \circ_1 f_2 \circ_1 \ldots \circ_1 f_n$ into 2-cells in $\mathcal{C}$ of length 1, such that each composite $f_i \circ_1 f_{i+1}$ is one of the following forms:



The two examples above illustrate how the stratified normalisation completion procedure can improve efficiency by decomposing the set of rules and analysing confluence stratum by stratum, thereby avoiding a global analysis of all critical branchings. In fact, the second example corresponds to a subcategory of the q-Schur category. In [9], we construct a 5-StRS presenting the q-Schur category and compute its normalisation and linear hom-basis, illustrating the effectiveness of our approach.

4 Conclusion

We have introduced a stratified normalisation completion procedure for linear rewriting, relying on a monomial order to orient newly generated rules. However, some rewriting systems do not admit any compatible monomial order (e.g. $xyz \to x^3+y^3+z^3$ [10]), while they may still be proved terminating by means of termination functions. This suggests developing a stratified normalisation procedure based on termination functions, independently of monomial orders.

Our constructions, formulated for rewriting in free linear precategories, could also be extended to the non-free setting by working modulo relations. This would require suitable stratifications, normalisation procedures, and termination functions for rewriting modulo.

Finally, the stratified normalisation procedure requires exploring a set of admissible contexts which is, in general, infinite and difficult to describe in higher dimensions. Two directions could be pursued to make the procedure more effective. The first consists of stratifying admissible contexts by their length, thereby yielding finite approximations, as initiated in Proposition 1. The second consists of describing admissible contexts by means of automata, as in the case of operated algebras [16, 17], in order to obtain a more efficient representation. These developments could also provide a basis for future implementations of the stratified normalisation procedure in computer algebra systems. Such implementations would be particularly valuable for diagrammatic algebras, which are often presented by many rewriting rules.

References

[1] Clément Alleaume. Rewriting in higher dimensional linear categories and application to the affine oriented Brauer category. *J. Pure Appl. Algebra*, 222(3):636–673, 2018.

[2] David J. Anick. On the homology of associative algebras. *Trans. Amer. Math. Soc.*, 296(2):641–659, 1986.

[3] Dimitri Ara, Albert Burroni, Yves Guiraud, Philippe Malbos, François Métayer, and Samuel Mimram. *Polygraphs: From Rewriting to Higher Categories*, volume 495 of *London Mathematical Society Lecture Note Series*. 2025. arXiv:2312.00429.

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

[5] Bruno Buchberger. History and basic features of the critical-pair/completion procedure. *J. Symbolic Comput.*, 3(1-2):3–38, 1987. Rewriting techniques and applications (Dijon, 1985).

[6] Benjamin Dupont. Rewriting modulo isotopies in Khovanov-Lauda-Rouquier's categorification of quantum groups. *Adv. Math.*, 378:Paper No. 107524, 75, 2021.

[7] Simon Forest and Samuel Mimram. Rewriting in gray categories with applications to coherence. *Mathematical Structures in Computer Science*, 32(5):574–647, 2022.

[8] Stéphane Gaussent, Yves Guiraud, and Philippe Malbos. Coherent presentations of Artin monoids. *Compos. Math.*, 151(5):957–998, 2015.

[9] Stéphane Gaussent, Zuan Liu, and Philippe Malbos. Diagrammatic hom-bases by confluence and normalisation. Preprint, 2026.

[10] Yves Guiraud, Eric Hoffbeck, and Philippe Malbos. Convergent presentations and polygraphic resolutions of associative algebras. *Math. Z.*, 293(1-2):113–179, 2019.

[11] Yves Guiraud and Philippe Malbos. Higher-dimensional normalisation strategies for acyclicity. *Adv. Math.*, 231(3-4):2294–2351, 2012.

[12] Gérard Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *J. Assoc. Comput. Mach.*, 27(4):797–821, 1980.

[13] Jean-Pierre Jouannaud and Helene Kirchner. Completion of a set of rules modulo a set of equations. In *Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '84, pages 83–92, New York, NY, USA, 1984. ACM.

[14] Jean-Pierre Jouannaud and Jianqi Li. Church-Rosser properties of normal rewriting. In *Computer science logic 2012*, volume 16 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 350–365. 2012.

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

[16] Zuan Liu and Philippe Malbos. Polygraphic resolutions for operated algebras. Preprint arXiv:2502.16304, 2025.

[17] Zuan Liu, Zihao Qi, Yufei Qin, and Guodong Zhou. Gröbner-shirshov bases for free multi-operated algebras over algebras. *Journal of Symbolic Computation*, 133:102489, 2026.

[18] Claude Marché. Normalized rewriting: an alternative to rewriting modulo a set of equations. *J. Symbolic Comput.*, 21(3):253–288, 1996.

[19] Gerald E. Peterson and Mark E. Stickel. Complete sets of reductions for some equational theories. *J. Assoc. Comput. Mach.*, 28(2):233–264, 1981.

[20] Léo Schelstraete. Rewriting modulo in diagrammatic algebras and application to categorification. Preprint, arXiv:2502.03028, 2025.

[21] Anatoly I. Shirshov. Some algorithmic problems for Lie algebras. *Sib. Mat. Zh.*, 3:292–296, 1962.

[22] Craig C. Squier. Word problems and a homological finiteness condition for monoids. *J. Pure Appl. Algebra*, 49(1-2):201–217, 1987.

[23] Craig C. Squier, Friedrich Otto, and Yuji Kobayashi. A finiteness condition for rewriting systems. *Theoret. Comput. Sci.*, 131(2):271–294, 1994.

[24] Victor A. Ufnarovskij. Combinatorial and asymptotic methods in algebra. In *Algebra, VI*, volume 57 of *Encyclopaedia Math. Sci.*, pages 1–196. Springer, Berlin, 1995.

Completion to Strong Confluence*

Salvador Lucas and Julia Pagan

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

Abstract

We describe *completion* procedures to obtain *strongly confluent* TRSs from a set of equations. Strong confluence implies confluence *without requiring termination*. We show the performance of our implementation in TRS.Tool 2 by means of some benchmarks.

1 Introduction

Confluence (Figure 1, left) and termination (the absence of infinite rewrite sequences $t_1 \to t_2 \to \cdots$) are usual requirements for rewriting-based equational reasoning (see [3, Chapter 4] to find missing definitions and notions we silently use in the following). In this setting, the process to obtain a *confluent and terminating* Term Rewriting System (TRS) $\mathcal{R}_E$ out from a set of equations E was investigated by Knuth and Bendix [8], and it is usually known as *completion*. During Knuth and Bendix' completion, given (i) a *reduction ordering* $\succ$, (ii) a TRS $\mathcal{R}_0$ (obtained from E) whose rules are all *compatible* with $\succ$ (i.e., $\ell \succ r$ for all $\ell \to r \in \mathcal{R}_0$ so that it is *terminating* [3, Theorem 5.2.3]), and (iii) a *non-*$\mathcal{R}_0$-joinable critical pair $\pi : \langle s, t\rangle$ of $\mathcal{R}_0$ (i.e., there is no u such that $s \to^*_{\mathcal{R}_0} u \,{}^*_{\mathcal{R}_0}\!\leftarrow t$ holds), we add a *new rule* $\alpha : \widehat{s} \to \widehat{t}$ (if $\widehat{s} \succ \widehat{t}$ holds) or $\alpha : \widehat{t} \to \widehat{s}$ (if $\widehat{t} \succ \widehat{s}$ holds; or *fail*, otherwise) to $\mathcal{R}_0$. Here $\widehat{s}$ and $\widehat{t}$ are $\mathcal{R}_0$-normal forms of s and t, respectively (computable as $\mathcal{R}_0$ is terminating). Then, we obtain a new TRS $\mathcal{R}_1 = \mathcal{R}_0 \cup \{\alpha\}$ where π is now $\mathcal{R}_1$-joinable, as $s \to^*_{\mathcal{R}_0} \widehat{s} \to_\alpha \widehat{t} \,{}^*_{\mathcal{R}_0}\!\leftarrow t$. Furthermore, $\mathcal{R}_1$ is terminating, as its rules are all compatible with $\succ$. We continue this way until (hopefully) obtaining a TRS $\mathcal{R}_n$ whose critical pairs are *all* $\mathcal{R}_n$-joinable. Then, by [7, Lemma 3.1], $\mathcal{R}_n$ is *locally confluent* (Figure 1, middle). Furthermore, since $\mathcal{R}_n$ is terminating by construction ($\mathsf{SN}(\mathcal{R}_n)$ for short), it is *confluent*, as $\mathsf{SN}(\to) \wedge \mathsf{WCR}(\to) \Rightarrow \mathsf{CR}(\to)$ holds for binary relations $\to$. However, it is well-known that this may *fail* in early stages such as (ii) above.

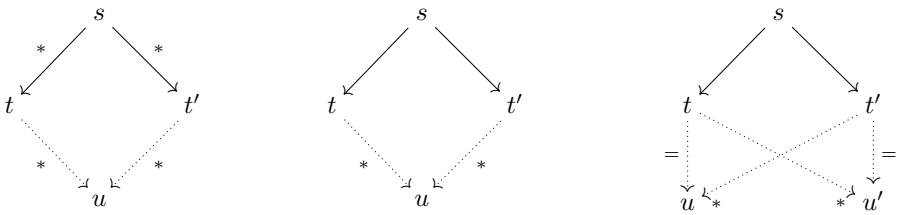
Example 1. *Consider* $E = \{\mathsf{c}(\mathsf{h}(\mathsf{b},\mathsf{b})) = \mathsf{h}(\mathsf{b},x), \mathsf{f}(\mathsf{f}(x)) = \mathsf{b}\}$. *The only TRS* $\mathcal{R}_0$ *which can be obtained from* E *is*

$$\mathsf{h}(\mathsf{b},x) \to \mathsf{c}(\mathsf{h}(\mathsf{b},\mathsf{b})) \quad (1)$$
$$\mathsf{f}(\mathsf{f}(x)) \to \mathsf{b} \quad (2)$$

which is not *terminating, as no reduction ordering* $\succ$ *is compatible with its rules.*

We have recently investigated rewriting-based techiques for proving equational goals, where termination of $\mathcal{R}_E$ is *not* required [11]. Since confluence can also be guaranteed *without requiring termination*, devising completion procedures which do not depend on termination is also interesting. In [10] we show how to do it by relying on *strong confluence* [7]. Strong confluence of a binary relation $\to$ ($\mathsf{SCR}(\to)$, see the third diagram in Figure 1, where $s \to^= t$

*Partially supported by MCIN/AEI project PID2024-162030OB-100 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe" and by the grant CIPROM/2022/6 funded by Generalitat Valenciana.



Confluence (CR) Local confluence (WCR) Strong confluence (SCR)

Figure 1: Different confluence properties

is $s \to t$ or $s = t$), implies confluence, i.e., $\mathsf{SCR}(\to) \Rightarrow \mathsf{CR}(\to)$ holds [7, Lemma 2.5]. Also strong confluence of *parallel reduction* $\Vdash$ (where *disjoint redexes* in terms s are *simultaneously* rewritten [7, page 814]) implies confluence of $\to$, i.e., $\mathsf{SCR}(\Vdash) \Rightarrow \mathsf{CR}(\to)$ cf. [7, page 815]. Furthermore, both $\mathsf{SCR}(\to)$ and $\mathsf{SCR}(\Vdash)$ can be proved by using alternative joinability notions (strong joinability, parallel-closedness [7]) with critical pairs. Given a set E of equations we show how to (hopefully) obtain a TRS $\mathcal{R}_E$ such that $\mathsf{SCR}(\to_{\mathcal{R}_E})$ or $\mathsf{SCR}(\Vdash_{\mathcal{R}_E})$ holds, thus guaranteeing confluence of $\mathcal{R}_E$, i.e., $\mathsf{CR}(\to_{\mathcal{R}_E})$, without relying on any reduction ordering $\succ$ to decide how equations in E become rules. Instead, we choose the orientation of the equations fulfilling the usual syntactic requirements of rewrite rules (the left-hand side is not a variable and variables in the right-hand side occur in the left-hand side) and also the conditions to apply the underlying theorems guaranteeing strong confluence of $\to$ or $\Vdash$ as some kind of joinability of critical pairs (linearity or left-linearity).

Example 2. *The critical pair* $\pi_{(2),1,(2)} : \langle \mathsf{f}(\mathsf{b}), \mathsf{b} \rangle$, *of* $\mathcal{R}$ *in Example 1 is* not *joinable. By adding*

$$\mathsf{f}(\mathsf{b}) \quad \to \quad \mathsf{b} \tag{3}$$

to obtain $\mathcal{R}_1 = \mathcal{R}_0 \cup \{(3)\}$, $\pi_{(2),1,(2)}$ *becomes* strongly joinable *using* $\mathcal{R}_1$. *The new CP* $\pi_{(2),1,(3)}$ *and the previous one* $\pi_{(2),1,(2)}$ *coincide. There is no other CP. Thus,* completion to strong confluence *finishes with* $\mathcal{R}_E = \{(1), (2), (3)\}$. EqValCSR[1] *[11] can now be used with* $\mathcal{R}_E$ *to prove* validity *of* $\mathsf{h}(\mathsf{f}(\mathsf{b}), \mathsf{f}(\mathsf{b})) = \mathsf{c}(\mathsf{c}(\mathsf{h}(\mathsf{f}(\mathsf{f}(x)), \mathsf{b})))$ *in* E *and also to* disprove *it for* $\mathsf{b} = \mathsf{h}(\mathsf{b}, \mathsf{b})$, *see [11, Examples 5 & 6]*

Remark 3 (Completion to Strong Confluence)**.** *In [8, Section 6], Knuth and Bendix talk about* "extension to a complete set" *to describe what is called* completion *today, but "complete" is used as an adjective. In [7, page 819], Huet uses "to complete" as an action (verb) and wrote*

> *it is possible to extend the Knuth and Bendix method, to attempt to complete a theory to a confluent one.*

In this paper, we use "completion" in this specific sense of an "attempt to complete a theory to a confluent one". Actually, Knuth and Bendix' reason for doing an "extension" is achieving local confluence as joinability of critical pairs, as termination is first assumed and then kept after the "extension" with new rules. Thus, the purpose of the procedure is not reinforcing termination but achieving confluence. Thus, we use completion *in the sense of extending/adding new rules to reinforce (strong) confluence of a TRS* $\mathcal{R}_E$ *corresponding an equational system* E.

After some preliminaries in Section 2, Section 3 recalls the notion of strong confluence and the results underlying our completion methods. Section 4 discusses completion to strong confluence of $\to$ and $\Vdash$. Section 5 briefly describes our benchmarks. Section 6 concludes.

[1] Available here: http://zenon.dsic.upv.es/trstool/EQ.aspx

2 Preliminaries

A (valid) *rewrite rule* (with label α) is an ordered pair $\alpha : \ell \to r$ or just $\ell \to r$, with $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\ell \notin \mathcal{X}$ and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell)$. The *left-hand side* (*lhs*) of the rule is ℓ and r is the *right-hand side* (*rhs*). A *Term Rewriting System* (TRS) is a pair $\mathcal{R} = (\mathcal{F}, R)$ where R is a set of rewrite rules. A rule $\ell \to r$ is *left-linear* if ℓ is a linear term; it is *linear* if r also is. A TRS $\mathcal{R}$ is *left-linear* (resp. linear) if all rules are left-linear (resp. linear). A term $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ rewrites to t at position p, written $s \xrightarrow{p}_{\mathcal{R}} t$ (or just $s \to t$), if $s|_p = \sigma(\ell)$ and $t = s[\sigma(r)]_p$, for some rule $\ell \to r$ in $\mathcal{R}$, $p \in \mathcal{P}os(s)$ and substitution σ. Terms s and t are *$\downarrow_{\mathcal{R}}$-joinable* (or just joinable if no confusion arises), written $s \downarrow_{\mathcal{R}} t$ if there is a term u such that $s \to^*_{\mathcal{R}} u$ and $t \to^*_{\mathcal{R}} u$. A TRS $\mathcal{R}$ is *confluent* (resp. *terminating*) if $\to_{\mathcal{R}}$ is confluent (resp. terminating). Given terms s and t, and a set $P = \{p_1, \ldots, p_n\} \subseteq \mathcal{P}os(s)$ of disjoint positions in s, $s \xrightarrow{P}_{\mathcal{R}} t$ is a *parallel reduction step* from s to t if there are rules $\alpha_i : \ell_i \to r_i \in \mathcal{R}$ and substitutions σ_i, $1 \le i \le n$, such that $s|_{p_i} = \sigma_i(\ell_i)$ and $t = s[\sigma_1(r_1)]_{p_1} \cdots [\sigma_n(r_n)]_{p_n}$. We often write $s \xrightarrow{P} t$, $s \Vdash_{\mathcal{R}} t$, or even $s \Vdash t$.

3 Confluence of Linear and Left-linear TRSs

Strong confluence corresponds to commutation of the rightmost diagram in Figure 1 (from [7, Figure 7]). Accordingly, s and t are *strongly $\to$-joinable* (or just *strongly joinable* if no confusion arises), if there are terms u, u' such that $s \to^= u \ {}^*\!\leftarrow t$ and $s \to^* u' \ {}^=\!\leftarrow t$.

In [7], strong confluence of TRSs is investigated using *critical pairs*. Given variable disjoint rules $\alpha : \ell \to r$ and $\alpha' : \ell' \to r'$ of a TRS $\mathcal{R}$, and a non-variable position p of ℓ such that $\ell|_p$ and ℓ' unify with *mgu* θ, $\langle \theta(\ell[r']_p), \theta(r) \rangle$, labeled $\pi_{\alpha,p,\alpha'}$, is a *critical pair* (CP) of $\mathcal{R}$ with *critical position* p. Let $\mathsf{CP}(\mathcal{R})$ be the set of CPs of $\mathcal{R}$. Critical pairs $\pi_{\alpha,p,\alpha'}$ whose critical position is the *root* position, i.e., $p = \Lambda$, are called *root critical pairs* in [3, Exercise 6.19].

Theorem 4. *Let $\mathcal{R}$ be a TRS. Then, (i) $\mathsf{SCR}(\to_{\mathcal{R}})$ holds if $\mathcal{R}$ is linear and all CPs are strongly $\to_{\mathcal{R}}$-joinable [7, Lemma 3.2]; and (ii) $\mathsf{SCR}(\Vdash_{\mathcal{R}})$ holds if $\mathcal{R}$ is left-linear and for all CPs $\pi_{\alpha,p,\alpha'} : \langle s, t \rangle \in \mathsf{CP}(\mathcal{R})$, if $p = \Lambda$, then s and t are strongly $\Vdash_{\mathcal{R}}$-joinable; and if $p \neq \Lambda$, then $s \Vdash_{\mathcal{R}} t$ (*parallel-closedness*) [13, Corollary 3.2].*

Although linear TRSs are left-linear, sometimes Theorem 4(i) applies when Theorem 4(ii) does *not* apply, see [10, Section 3]. Thus, we use *both* in our completion procedure.

A sequence of atoms A_i, denoting rewriting conditions $s_i \to^* t_i$, $s_i \to^= t_i$, $s \Vdash t$, ..., for $1 \le i \le n$ is *feasible* if there is a substitution σ such that $\sigma(A_i)$ holds for all $1 \le i \le n$; otherwise, it is *infeasible* [9]. Also, as it is well-known, in term rewriting the variables occurring in subject terms behave as *constants* during rewriting sequences. Thus, given a term t, we let $t^\downarrow$ be the ground term obtained by replacing each occurrence of a variable x in t by a fresh constant c_x (see, e.g., [2]). For a TRS $\mathcal{R}$, terms s and t, and variables u and u', $\langle s, t \rangle$ is

$$\begin{array}{rl} \textit{strongly } \to_{\mathcal{R}}\textit{-joinable} & \text{iff } s^\downarrow \to^= u,\ s^\downarrow \to^* u',\ t^\downarrow \to^* u,\ t^\downarrow \to^= u' \quad \text{is feasible} \\ \textit{strongly } \Vdash_{\mathcal{R}}\textit{-joinable} & \text{iff } s^\downarrow \Vdash u,\ s^\downarrow \to^* u',\ t^\downarrow \to^* u,\ t^\downarrow \Vdash u' \quad \text{is feasible} \\ \textit{parallel closed} & \text{iff } s^\downarrow \Vdash t^\downarrow \quad \text{is feasible} \end{array}$$

We use infChecker [5] for proving and disproving these feasibility conditions.

Example 5. *Strong joinability of $\langle \mathsf{f}(\mathsf{b}), \mathsf{b} \rangle$ in Example 1 is* disproved *with* infChecker *by uploading a COPS infeasibility problem with the* condition *component*

```
(VAR u u')
(CONDITION  f(b) ->= u, f(b) ->* u', b ->* u, b ->= u'   )
```

Algorithm 1: Completion to Linear, Strongly $\to$-Confluent TRSs

$i := 0$; $R_0 := \{\ell \to r \mid (\ell \approx r) \in E \cup E^{-1} \wedge \ell \notin \mathcal{X} \wedge \mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell)\}$; $oldCPs_0 := \{\}$
repeat
  $R_{i+1} := R_i$;
  $oldCPs_{i+1} := oldCPs_i$;
  for all $\langle s,t\rangle \in (CP(R_i) - oldCPs_i)$ **do**
    if $SJoin(\langle s,t\rangle) =$ `False` **or** $SJoin(\langle s,t\rangle) =$ `Maybe` **then**
      if $s \notin \mathcal{X} \wedge \mathcal{V}ar(t) \subseteq \mathcal{V}ar(s)$ **then**
        $R_{i+1} := R_{i+1} \cup \{s \to t\}$;
      else
        if $t \notin \mathcal{X} \wedge \mathcal{V}ar(s) \subseteq \mathcal{V}ar(t)$ **then**
          $R_{i+1} := R_{i+1} \cup \{t \to s\}$;
        else
          Terminate with output `Fail`;
        end
      end
    end
    $oldCPs_{i+1} := oldCPs_{i+1} \cup \{\langle s,t\rangle, \langle t,s\rangle\}$;
  od
  $i := i + 1$;
until $R_i = R_{i-1}$;
output R_i;

Handling feasibility conditions with goals $s \Vdash t$ is more involved, but see [10, Section 4].

4 Completion to Strong Confluence

In our completion approach, we deal with CPs $\langle s,t\rangle$ using Theorems 4(i) and (ii) as follows:

1. Theorem 4(i): if $\langle s,t\rangle$ is not strongly $\to_{\mathcal{R}}$-joinable, add rule $s \to t$, if valid, or else $t \to s$, if valid, or fail otherwise (see Example 6 below).

2. Theorem 4(ii): if $\langle s,t\rangle$ is a root CP and it is not strongly $\Vdash_{\mathcal{R}}$-joinable, then add rule $s \to t$, if valid; or else $t \to s$, if valid; or fail otherwise. If π is not a root CP and it is not parallel-closed, then add $s \to t$, if valid; or *fail* otherwise, as $t \to s$ does *not* work, being parallel closedness an *asymmetric* joinability notion.

Given a set of equations E, Algorithm 1 tries to find a linear, strongly $\to$-confluent TRS $\mathcal{R}_E$ using Theorem 4(i). Furthermore, the equational theories of $E_{\mathcal{R}_E} = \{\ell = r \mid \ell \to r \in \mathcal{R}_E\}$ and E coincide. The algorithm begins with a finite set of equations E satisfying input requirements guaranteeing that a first set of rules R_0 can be obtained by orienting each equation $s = t$ as $s \to t$ if $s \notin \mathcal{X}$ and $\mathcal{V}ar(t) \subseteq \mathcal{V}ar(s)$; otherwise, $t \to s$ is used. For each critical pair $\pi : \langle s,t\rangle \in \mathsf{CP}(\mathcal{R}_i)$, the algorithm checks its strong $\to_{\mathcal{R}_i}$-joinability using infChecker. If (i) strong joinability of π is *disproved* or (ii) the tool *fails* to determine strong joinability, then either $s \to t$ or $t \to s$ is added as a new rule provided that in one of the two cases the left-hand side is not a variable and the variables in the right-hand side already occur in the left-hand side. Otherwise, the algorithm *fails*. This may happen.

Table 1: Completion to Strong Confluence: Experiments

		Added Rules												
	Success	**0**	**1**	**2**	**3**	**4**	**6**	**8**	**9**	**11**	**12**	**No Result**	**Fail**	**Total**
Alg. 1	39 (33%)	0	19	8	5	3	2	1	0	1	0	79	0	118
Alg. 2	13 (27%)	1	3	2	2	1	2	0	1	0	1	31	2	46

Example 6. *The linear TRS* $\{\mathsf{f}(\mathsf{g}(x,y)) \to y, \mathsf{g}(x,y) \to x\}$ *has a non-joinable critical pair* $\langle \mathsf{f}(x), y\rangle$ *which cannot be oriented into a valid rewrite rule.*

On the other hand, since both s and t are guaranteed to be linear terms, the added rule is linear.

If one of the rules can be added, strong joinability of π is trivially guaranteed. Critical pairs that have already been processed in previous iterations of the algorithm are skipped. We do so by keeping track of the analysed pairs in $oldCPs_i$, which collects the already considered critical pairs for rules in previous TRSs, $\mathcal{R}_0, \mathcal{R}_1, \ldots, \mathcal{R}_{i-1}$. The process is repeated until no new rule is added. If the algorithm successfully finishes, then the output is a (linear) strongly confluent TRS $\mathcal{R}_E$. If some critical pair cannot be oriented (see, e.g., Example 6), then the algorithm stops with *failure*. If the loop is executed infinitely, each iteration producing non strongly $\to$-joinable CPs, never meeting the ending condition, $\mathcal{R}_i = \mathcal{R}_{i-1}$, then, there is no explicit, finitely reported failure message. For this reason we talk of an *infinite failure*.

Theorem 7. *[12] Let E be a finite set of linear equations satisfying the input requirements in Algorithm 1. If the completion procedure applied to E terminates successfully with output $\mathcal{R}_n$, then $\mathcal{R}_n$ is a finite strongly confluent TRS and the equational theories of $E_{\mathcal{R}_n}$ and E coincide.*

Algorithm 2 tries to find a left-linear strongly $\Vdash$-confluent TRS $\mathcal{R}_E$ using Theorem 4(ii). By lack of space we cannot display it here. See [10, Section 5.2] for full details.

5 Implementation and experiments

Our algorithms have been implemented on top of the tool TRS.Tool [1], which is written in C#, on the .NET platfom, thus producing a new, improved and extended version of the tool called TRS.Tool 2 , http://zenon.dsic.upv.es/trstool/. For our *benchmarks*,

http://zenon.dsic.upv.es/trstool/benchmarks/CompSCR/

we used *experimental datasets* obtained from the *Confluence Problems Database (COPS)* [6]. Sets of equations $E_{\mathcal{R}} = \{\ell = r \mid \ell \to r \in \mathcal{R}\}$ were obtained from TRSs $\mathcal{R}$ from COPS.

Table 1 summarizes our results by including (i) the number of successfully completed TRSs, (ii) the number of examples that included a given number of additional rules, (iii) the number of examples whose completion was interrupted due to timeout (15 minutes), (iv) the number of failing completions, and (v) the total number of considered examples. As for Algorithm 1, COPS *search* sequence

```
linear non_terminating non_confluent
```

extracts 118 linear, nonterminating, and nonconfluent TRSs to be used in the benchmarks. As

shown in Table 1, we had 33% of successful completions. No *failure* was registered. A timeout interrupted the remaining ones. As for Algorithm 2, the *search* sequence

```
left_linear non_linear non_terminating non_weakly_orthogonal
```

extracts 46 left-linear but not linear, non-terminating, and non-weakly orthogonal TRSs to be used in the benchmarks. Note that `non_confluent` is *not* included among the *search* words used to extract the examples. The reason is that only 5 examples are obtained in this way: 49, 64, 113, 584, and 762. This is too small a set of benchmarks to test the algorithm. The use of `non_weakly_orthogonal` avoids many strongly confluent systems, except `504`, i.e.,

```
(VAR x y z)
(RULES
  +(x,y) -> +(y,x)
  *(+(x,y),z) -> +(*(x,z),*(y,z))
  *(+(y,x),z) -> +(*(x,z),*(y,z))
)
```

which is left-linear but not weakly orthogonal, and is proved strongly $\Vdash$-confluent using Toyama's result (the three CPs are root CPs and strongly joinable) and corresponds to the entry with 0 added rules in the second row of Table 1, for Algorithm 2. Still, Algorithm 2 was able to obtain a strongly $\Vdash$-confluent TRS for the non-confluent (according to COPS) TRSs 49, 64, 113, and 762, whereas completion of 584 was interrupted due to the timeout. As shown in Table 1, we had 27% of successful completions. The completion *failed* in 2 cases (examples 203 and 208). The combined use of Algorithms 1 and 2 provides a completion method which we call CompSCR `http://zenon.dsic.upv.es/trstool/CompSCR.aspx`: if E is *not* a set of linear equations, Algorithm 2 is used; otherwise, use Algorithm 1; if it fails or exceeds a timeout use Algorithm 2.

6 Conclusions and Future Work

We have introduced new procedures for completion of a set E of equations into a confluent (possibly nonterminating) TRS $\mathcal{R}_E$ using different kind of *strong confluence* properties so that $\leftrightarrow^*_{\mathcal{R}_E}$ is the equational theory of E. As far as we know, the idea of *completion to strong confluence* and its implementation in CompSCR are novel contributions in the field. In the near future, we plan to improve the efficiency of our implementation and explore other completion approaches based on exploiting other joinability notions for critical pairs which do not require termination. For instance, Gramlich's notion of *parallel critical pair* [4] can be considered for this purpose.

Acknoledgements. We thank the reviewers for their remarks and comments leading to several improvements.

References

[1] Salvador Arnal. Term Rewriting Systems .Net Framework (master thesis). Master's thesis, Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València, Spain, July 2013.

[2] Jürgen Avenhaus and Carlos Loría-Sáenz. On Conditional Rewrite Systems with Extra Variables and Deterministic Logic Programs. In Frank Pfenning, editor, *Logic Programming and Automated Reasoning, 5th International Conference, LPAR'94, Proceedings*, volume 822 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 1994.

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

[4] Bernhard Gramlich. Confluence without Termination via Parallel Critical Pairs. In Hélène Kirchner, editor, *Trees in Algebra and Programming - CAAP'96, 21st International Colloquium, Proceedings*, volume 1059 of *Lecture Notes in Computer Science*, pages 211–225. Springer, 1996.

[5] Raúl Gutiérrez and Salvador Lucas. Proving and disproving feasibility with infChecker. In Raúl Gutérrez and Naoki Nishida, editors, *14th International Workshop on Confluence, IWC 2025*, pages 38–44, 2025.

[6] Nao Hirokawa, Julian Nagele, and Aart Middeldorp. Cops and CoCoWeb: Infrastructure for Confluence Tools. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 346–353. Springer, 2018.

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

[8] Donald E. Knuth and Peter E. Bendix. Simple Word Problems in Universal Algebra. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[9] Salvador Lucas and Raúl Gutiérrez. Use of logical models for proving infeasibility in term rewriting. *Inf. Process. Lett.*, 136:90–95, 2018.

[10] Salvador Lucas and Julia Pagan. Completion to strongly confluent term rewriting systems. In Fernando Sáenz-Pérez, editor, *Proceedings XXV Jornadas sobre Programación y Lenguajes, PROLE 2026, Alicante, Spain, 16-18th June 2026*, pages 1–21. Sistedes, 2026. handle: 11705/PROLE/2026/11.

[11] Salvador Lucas and Julia Pagan. Proving Equations with Nonterminating Term Rewriting Systems using Context-Sensitive Rewriting. In Silvio Ghilardi and Cleo Pau, editors, *Proceedings of the 40th International Workshop on Unification, UNIF 2026, Lisbon, Portugal, 24th July 2026*, volume to appear, 2026.

[12] Julia Pagan. Completion of Term Rewriting Systems to Ensure Strong Confluence (master thesis). Master's thesis, Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València, Spain, September 2025.

[13] Yoshihito Toyama. Commutativity of Term Rewriting Systems. In Kazuhiro Fuchi and Laurent Kott, editors, *Programming of Future Generation Computer, II. Proceedings of the Second Franco-Japanese Symposium on Programming of Future Generation Computers, Cannes, France, 9-11, November, 1987*, pages 393–407. North-Holland, 1988.

Confluence Competition 2026

Aart Middeldorp[1], Naoki Nishida[2], Teppei Saito[3],
René Thiemann[1], and Sarah Winkler[4]

[1] Department of Computer Science, University of Innsbruck, Austria
[2] Department of Computing and Software Systems, Nagoya University, Japan
[3] School of Information Science, JAIST, Japan
[4] Free University of Bozen–Bolzano, Italy

The next few pages in these proceedings contain the descriptions of the tools participating in the 15th Confluence Competition (CoCo 2026). 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 16 tools (listed in order of registration) participating in 11 categories (listed in order of first appearance in CoCo):

	TRS	CTRS	GCR	UNR	UNC	NFP	COM	INF	SRS	CSR	LCTRS
Natto								✓			
Hakusan	✓										
crest											✓
CeTA-Prover	✓						✓		✓		
CeTA	⋆						⋆	⋆	⋆		
CSI	✓			✓	✓	✓			✓		
CO3	✓	✓						✓			
CRaris											✓
FORT-h	✓		✓	✓	✓	✓	✓				
FORTify	⋆		⋆	⋆	⋆	⋆	⋆				
AProVE	✓										
CONFident	✓	✓							✓	✓	
infChecker								✓			
ACP	✓	✓		✓	✓	✓	✓		✓		
AGCP			✓								
nonreach+CeTA-Prover								✓			

CoCo adopts the ARI[1] format for input problems. Tools producing certifiable output in a specific category team up with a certifier and participate as combination in that category. The certifiers in CoCo 2026 are CeTA and FORTify. The latter teamed up with FORT-h. The former with ACP, CeTA-Prover, CSI and Hakusan in the TRS category, with ACP, CeTA-Prover and CSI in the SRS category, with ACP and CeTA-Prover in the COM category, and with Natto and nonreach+CeTA-Prover in the INF category.

[1] https://project-coco.uibk.ac.at/ARI/

The winning (for combined YES/NO answers) tools[2] of CoCo 2025 participated as demonstration tools, to provide a benchmark to measure progress.

The live run of CoCo 2026 on StarExec Miami [2] can be viewed at https://ari-cops.uibk.ac.at/liveview/?comp=CoCo.2026.competition. Further information about CoCo 2026, including a description of the categories and detailed results, can be obtained from https://project-coco.uibk.ac.at/2026/.

Acknowledgements The CoCo steering committee is grateful to Nao Hirokawa and Geoff Sutcliffe 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.

[2] They are not listed in the table but see http://project-coco.uibk.ac.at/2025/results.php.

Natto 0.2: An Infeasibility Prover

Teppei Saito

JAIST, Japan

Natto is a prototype tool for infeasibility analysis, primarily developed to assess the impact of formalizing order-based infeasibility methods for certification [3]. It implements a subset of the infeasibility techniques used in the Nagoya Termination Tool [1, 2], namely polynomial interpretations over the integers and the weighted path order.

Compared to the 2025 version, the weighted path order has been extended to the *generalized* weighted path order [4]. Proof output in the CFP 3 format is currently unsupported.

References

[1] A. Yamada, K. Kusakari, and T. Sakabe. Nagoya Termination Tool. In *Proc. of RTA-TLCA 2014*, LNCS 8560, pages 466–475, 2014.

[2] A. Yamada. Term orderings for non-reachability of (conditional) rewriting. In *Proc. of the 11th IJCAR*, LNAI 13385, pages 248–267, 2022.

[3] D. Kim, T. Saito, R. Thiemann, and A. Yamada. An Isabelle formalization of co-rewrite pairs for non-reachability in term rewriting. In *Proc. of the 14th CPP*, pages 272–282, 2025.

[4] T. Saito. Unifying semantic path order and weighted path order. PhD thesis, Japan Advanced Institute of Science and Technology, 2026.

Hakusan 0.13: A Confluence Tool

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

JAIST, Japan
hirokawa@jaist.ac.jp, s.kiraku@gmail.com

Hakusan is a confluence tool for left-linear term rewrite systems (TRSs). It analyzes confluence by successive application of *rule removal* criteria [3, 4, 8] based on rule labeling [7, 10], critical pair systems [2], and the generalization of Knuth and Bendix' criterion by Klein and Hirokawa [5]. Non-confluence is shown by tree automata techniques as CSI does [6]. Except for the generalization of Knuth and Bendix' criterion, Hakusan can produce proof certificates verifiable by CeTA [9], see [1]. The tool is available at the following website:

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

Note that this version is essentially the same as the version that participated in CoCo 2025.

References

[1] N. Hirokawa, D. Kim, K. Shintani, and R. Thiemann. Certification of confluence- and commutation-proofs via parallel critical pairs. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 147–161, 2024. doi:10.1145/3636501.3636949.

[2] N. Hirokawa and A. Middeldorp. Decreasing diagrams and relative termination. *Journal of Automated Reasoning*, 47:481–501, 2011. doi:10.1007/s10817-011-9238-x.

[3] N. Hirokawa and K. Shintani. Rule removal for confluence. In *Proceedings of 13th International Workshop on Confluence*, pages 50–54, 2024.

[4] F. Kawano, N. Hirokawa, and K. Shintani. Hakusan 0.11: A confluence tool. In *Proceedings of 13th International Workshop on Confluence*, pages 67–68, 2024.

[5] D. Klein and N. Hirokawa. Confluence of non-left-linear TRSs via relative termination. In *Proceedings of 18th International Conference on Logic Programming and Automated Reasoning*, volume 7180 of *LNCS*, pages 258–273, 2012. doi:10.1007/978-3-642-28717-6_21.

[6] J. Nagele, B. Felgenhauer, and A. Middeldorp. CSI: New evidence — a progress report. In *Proceedings of 26th International Conference on Automated Deduction*, pages 385–397, 2017. doi:10.1007/978-3-319-63046-5_24.

[7] V. van Oostrom. Confluence by decreasing diagrams, converted. In *Proceedings of 19th International Conference on Rewriting Techniques and Applications*, volume 5117 of *LNCS*, pages 306–320, 2008. doi:10.1007/978-3-540-70590-1_21.

[8] K. Shintani and N. Hirokawa. Compositional confluence criteria. *Logical Methods in Computer Science*, 20:6:1–6:28, 2024. doi:10.46298/lmcs-20(1:6)2024.

[9] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *LNCS*, pages 452–468, 2009.

[10] H. Zankl, B. Felgenhauer, and A. Middeldorp. Labelings for decreasing diagrams. *Journal of Automated Reasoning*, 54(2):101–133, 2015. doi:10.1007/s10817-014-9316-y.

CoCo 2026 Participant: crest 1.0

Jonas Schöpf[1] and Aart Middeldorp[2]

[1] Data Lab Hell, Zirl, Austria
jonas.schoepf@datalabhell.ac.at
[2] Department of Computer Science, University of Innsbruck, Austria
aart.middeldorp@uibk.ac.at

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

https://jonas.schoepf.me/crest

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

Our tool crest participates in the LCTRS category of CoCo 2026.

References

[1] Cynthia Kop. Termination of LCTRSs. *CoRR*, abs/1601.03206, 2016. doi: 10.48550/ARXIV.1601.03206.

[2] Cynthia Kop and Naoki Nishida. Term rewriting with logical constraints. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *Proc. 9th International Symposium on Frontiers of Combining Systems*, volume 8152 of *Lecture Notes in Artificial Intelligence*, pages 343–358, 2013. doi: 10.1007/978-3-642-40885-4_24.

[3] Julian Nagele, Bertram Felgenhauer, and Aart Middeldorp. Improving automatic confluence analysis of rewrite systems by redundant rules. In Maribel Fernández, editor, *Proc. 26th International Conference on Rewriting Techniques and Applications*, volume 36 of *Leibniz International Proceedings in Informatics*, pages 257–268, 2015. doi: 10.4230/LIPIcs.RTA.2015.257.

[4] Jonas Schöpf and Aart Middeldorp. Confluence criteria for logically constrained rewrite systems. In Brigitte Pientka and Cesare Tinelli, editors, *Proc. 29th International Conference on Automated Deduction*, volume 14132 of *Lecture Notes in Artificial Intelligence*, pages 474–490, 2023. doi: 10.1007/978-3-031-38499-8_27.

[5] Jonas Schöpf and Aart Middeldorp. Automated analysis of logically constrained rewrite systems using crest. In Arie Gurfinkel and Marijn Heule, editors, *Proc. 31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 15696 of *Lecture Notes in Computer Science*, pages 124–144, 2025. doi: 10.1007/978-3-031-90643-5_7.

[6] Jonas Schöpf and Aart Middeldorp. Improving confluence analysis for LCTRSs. In *Proc. 14th International Workshop on Confluence*, 2025. https://iwc2025.github.io/IWC2025_proceedings.pdf.

[7] Jonas Schöpf, Fabian Mitterwallner, and Aart Middeldorp. Confluence of logically constrained rewrite systems revisited. In Christoph Benzmüller, Marijn J.H. Heule, and Renate A. Schmidt, editors, *Proc. 12th International Joint Conference on Automated Reasoning*, volume 14740 of *Lecture Notes in Artificial Intelligence*, pages 298–316, 2024. doi: 10.1007/978-3-031-63501-4_16.

[1] https://ari-informatik.uibk.ac.at/

CoCo 2026 Participant: CeTA-Prover

René Thiemann

University of Innsbruck, Austria

The certifier CeTA [1] is a well-known participant of CoCo. In this year CeTA supports certain proofs, for which there were no certificate-producing tools available. This led to development of a new tool: CeTA-Prover. Its implementation consists of two parts.

First, several verified functions of CeTA are imported, e.g., verified algorithms for unification, for computing simultaneous critical pairs (SCPs), and for computing rewrite steps, i.e., single steps $\to$ and multisteps $\multimap$. Moreover, fast non-reachability criteria via *tcap* are imported.

Second, based on these verified functions, further unverified Haskell code has been added. At the core is a semi-decision procedure for the reachability problem: given a TRS and terms s and $t_1, \ldots, t_n$, does $s \to^* t_i$ hold for some t_i? In the affirmative case, the search algorithm also produces a witness in the form of a rewrite sequence. Internally, an iterative breadth-first search algorithm with pruning is encoded. Besides the core reachability prover, also some infrastructure is implemented for parsing ARI-files, for generating CPF-files, etc.

Wrapper functions around these algorithms then form the executable CeTA-Prover. It participates in three CoCo categories.

First, the reachability prover is combined with the SCP algorithm and the implementation of multistep rewriting in order to assemble a semi-decision procedure for *strong confluence* of $\multimap_{\mathcal{R}}$ for left-linear TRSs $\mathcal{R}$ via Okui's criterion [2].

Second, the same combination is used to assemble a semi-decision procedure for *strong commutation* of $\multimap_{\mathcal{R}}$ and $\multimap_{\mathcal{S}}$ for two left-linear TRSs $\mathcal{R}$ and $\mathcal{S}$ [3]. Since strong commutation of $\multimap_{\mathcal{R}}$ and $\multimap_{\mathcal{S}}$ is not equivalent to strong commutation of $\multimap_{\mathcal{S}}$ and $\multimap_{\mathcal{R}}$, both properties are tested in an interleaved setting, using the iterative interface of the breadth-first search.

Third, the reachability prover is coupled with the feasibility prover of nonreach [4] in the *infeasibility category* [5].

For further details and for accessing CeTA-Prover we refer to:

https://isafor-ceta.uibk.ac.at/ceta-prover.php

References

[1] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. *Proceedings of the International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009. doi:10.1007/978-3-642-03359-9_31.

[2] Satoshi Okui. Simultaneous critical pairs and Church-Rosser property. *Proceedings of the 9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *LNCS*, pages 2–16. Springer, 1998. doi:10.1007/BFB0052357.

[3] René Thiemann. Verifying and Generalizing Simultaneous Critical Pairs. *Proceedings of the 15th International Workshop on Confluence, July 24, 2026.* This volume.

[4] Florian Meßner and Christian Sternagel. nonreach – A Tool for Nonreachability Analysis. *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 11427 of *LNCS*, pages 337–343. Spring, 2019. URL: doi:10.1007/978-3-030-17462-0_19.

[5] Florian Meßner and René Thiemann. CoCo 2026 Participant: nonreach + CeTA-Prover. *Proceedings of the 15th International Workshop on Confluence, July 24, 2026.* This volume.

CoCo 2026 Participant: CeTA 3.9

Christina Kirk and René Thiemann

University of Innsbruck, Austria

The tool CeTA [1] is a certifier for, among other properties, (non-)confluence of term rewrite systems (TRSs) with and without conditions, (non-)commutation of TRSs and (in)feasibility of conditions w.r.t. some conditional TRSs. 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:

https://isafor-ceta.uibk.ac.at/

We here describe only the most relevant change of CeTA 3.9 in comparison to the version of the previous year: It is the full support of Okui's confluence criterion [2]. Previously, only the soundness statement was proven [3], without an algorithm to compute the set of simultaneous critical pairs (SCPs) for a TRS. In CeTA 3.9 such an algorithm is now implemented, verified, and used to to develop a certificate checker for Okui's criterion [4]. The checker expects certificates as follows, where rewrite sequences $\to^*$ can be represented succinctly via multisteps $\multimap^*$.

- For every non-trivial SCP $s \leftarrow\!\circ\!- \cdot \to t$ (i.e., $s \neq t$), a joining sequence needs to be provided, by writing down the intermediate terms $u_1, \ldots, u_n$ in the certificate.

$$s \multimap u_1 \multimap u_2 \multimap \ldots \multimap u_n \leftarrow\!\circ\!- t$$

CeTA also supports commutation proofs via a generalization of Okui's criterion [4]. Given two left-linear TRSs $\mathcal{R}$ and $\mathcal{S}$, in order to prove strong commutation of $\multimap_\mathcal{R}$ and $\multimap_\mathcal{S}$, we require certificates that have the same design as in the confluence setting.

- For every non-trivial SCP $s \,{}_\mathcal{R}\!\leftarrow\!\circ\!- \cdot \to_\mathcal{S} t$, a joining sequence needs to be provided.

$$s \multimap_\mathcal{S} u_1 \multimap_\mathcal{S} u_2 \multimap_\mathcal{S} \ldots \multimap_\mathcal{S} u_n \,{}_\mathcal{R}\!\leftarrow\!\circ\!- t$$

References

[1] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009. doi:10.1007/978-3-642-03359-9_31.

[2] Satoshi Okui. Simultaneous critical pairs and Church-Rosser property. In Tobias Nipkow, editor, *Rewriting Techniques and Applications, 9th International Conference, RTA-98, Tsukuba, Japan, March 30 - April 1, 1998, Proceedings*, volume 1379 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 1998. URL: https://doi.org/10.1007/BFb0052357, doi:10.1007/BFB0052357.

[3] Christina Kirk and Aart Middeldorp. Formalizing simultaneous critical pairs for confluence of left-linear rewrite systems. In Kathrin Stark, Amin Timany, Sandrine Blazy, and Nicolas Tabareau, editors, *Proceedings of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2025, Denver, CO, USA, January 20-21, 2025*, pages 156–170. ACM, 2025. doi:10.1145/3703595.3705881.

[4] René Thiemann. Verifying and Generalizing Simultaneous Critical Pairs. *Proceedings of the 15th International Workshop on Confluence, July 24, 2026.* This volume.

CoCo 2026 Participant: CSI 1.2.8

Aart Middeldorp and René Thiemann

Department of Computer Science, University of Innsbruck, Austria
aart.middeldorp@uibk.ac.at, rene.thiemann@uibk.ac.at

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

https://cl-informatik.uibk.ac.at/research/software/csi

under a LGPLv3 license. A detailed description of CSI can be found in [4]. Some of the implemented techniques are described in [1,3,6]. CSI can also produce certificates for confluence results, which are checked by CeTA [2].

CSI participates in the SRS and TRS categories of CoCo 2026 both as a standalone tool and in combination with CeTA providing certified confluence and non-confluence answers. In addition, CSI participates in the NFP, UNC and UNR categories. In 2025 CSI won not only the TRS category, but also with CeTA it won the RELIABILITY category where tools are ranked based on the number of certifiable answers.

There is one novelty about CSI in 2026: whereas Okui's confluence criterion [5] was previously only used in the unrestricted mode of CSI, now CSI also uses this criterion in the certified mode with proper CPF output.

CSI uses the conversion tool[1] to transform ARI problems into COPS problems.

References

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

[2] Christina Kirk and René Thiemann. CoCo 2026 Participant: CeTA 3.9. In *Proc. 15th International Workshop on Confluence*, 2026. This volume.

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

[4] Julian Nagele, Bertram Felgenhauer, and Aart Middeldorp. CSI: New Evidence – A Progress Report. In *Proc. 26th International Conference on Automated Deduction*, volume 10395 of *LNAI*, pages 385–397, 2017. doi: 10.1007/978-3-319-63046-5_24.

[5] Satoshi Okui. Simultaneous critical pairs and Church-Rosser property. *Proc. 9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *LNCS*, pages 2–16, 1998. doi: 10.1007/BFB0052357.

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

[1] https://project-coco.uibk.ac.at/ARI/#conversion

CO3 (Version 2.7)

Naoki Nishida and Misaki Kojima

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

CO3, a **CO**nverter for proving **CO**nfluence of **CO**nditional 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 $\mathbb{U}_{conf}$ [2], a variant of the *unraveling* $\mathbb{U}$ [9], and then verifies confluence of the transformed TRS by using the following theorem.

Theorem 1 ([1, 2])**.** *A 3-DCTRS $\mathcal{R}$ is confluent if $\mathcal{R}$ is WLL and $\mathbb{U}_{conf}(\mathcal{R})$ is confluent.*

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

The difference from the previous version [6] is a light-weight implementation of LPO and RPO used in the reduction pair processor for termination. Since the main goal of CO3 is to provide a converter of CTRSs to TRSs, CO3 has maintained a design policy that does not rely on external tools. For this reason, CO3 does not call any SAT solver and uses the precedence determined as follows: Any defined symbol is greater than any constructor, and regarding symbols of the same type (defined symbols or constructors), those declared earlier are greater than those declared later. Unfortunately, the implementation of LPO and RPO has not yielded any new successful (dis)proofs regarding CTRSs in the ARI database, while confluence of `1498.ari` has newly been proved. For this reason, we register for the `TRS` category this year.

References

[1] K. Gmeiner, B. Gramlich, and F. Schernhammer. On soundness conditions for unraveling deterministic conditional rewrite systems. In A. Tiwari, editor, *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications*, volume 15 of *Leibniz International Proceedings in Informatics*, pages 193–208. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

[2] K. Gmeiner, N. Nishida, and B. Gramlich. Proving confluence of conditional term rewriting systems via unravelings. In N. Hirokawa and V. van Oostrom, editors, *Proceedings of the 2nd International Workshop on Confluence*, pages 35–39, 2013.

[3] Y. Maeda, N. Nishida, M. Sakai, and T. Kobayashi. Extending narrowing trees to basic narrowing in term rewriting. IEICE Technical Report SS2018-39, the Institute of Electronics, Information and Communication Engineers, 2019. Vol. 118, No. 385, pp. 73–78, in Japanese.

[1] http://www.lctrs.jp/tools/co3/

[4] N. Nishida. CO3 (Version 2.1). In M. Ayala-Rincón and S. Mimram, editors, *Proceedings of the 9th International Workshop on Confluence*, page 67, 2020.

[5] N. Nishida. CO3 (Version 2.2). In S. Mimram and C. Rocha, editors, *Proceedings of the 10th International Workshop on Confluence*, page 61, 2021.

[6] N. Nishida and M. Kojima. CO3 (Version 2.6). In R. Gutiérrez and N. Nishida, editors, *Proceedings of the 14th International Workshop on Confluence*, pages 65–66, 2025.

[7] N. Nishida, T. Kuroda, and K. Gmeiner. CO3 (Version 1.3). In B. Accattoli and A. Tiwari, editors, *Proceedings of the 5th International Workshop on Confluence*, page 74, 2016.

[8] N. Nishida and Y. Maeda. Narrowing trees for syntactically deterministic conditional term rewriting systems. In H. Kirchner, editor, *Proceedings of the 3rd International Conference on Formal Structures for Computation and Deduction*, volume 108 of *Leibniz International Proceedings in Informatics*, pages 26:1–26:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

[9] E. Ohlebusch. Termination of logic programs: Transformational methods revisited. *Applicable Algebra in Engineering, Communication and Computing*, 12(1/2):73–116, 2001.

CRaris (Version 1.2)

Naoki Nishida and Misaki Kojima

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

CRaris, a **CR** checker for LCTRSs in **ari** style,[1] is a tool to prove confluence of *logically constrained term rewrite systems* (LCTRSs, for short) [4] written in the ARI format [1].[2] The tool is based on Crisys2[3]—**C**onstrained **r**ewriting **i**nduction **sys**tem (version **2**)—and receives LCTRSs written in the ARI format only to prove confluence, while Crisys2 has many functions to, e.g., solve *all-path reachability problems* [2]. To prove confluence of LCTRSs, the tool uses the following criteria:

- weak orthogonality [4], and
- termination and joinability of critical pairs [7].

To prove termination, the tool uses the DP framework for LCTRSs [3] without any interpretation method, together with a criterion for LCTRSs with bitvector arithmetics [5].

The *critical pairs* of two constrained rewrite rules $\rho_1 : \ell_1 \to r_1\ [\varphi_1]$ and $\rho_2 : \ell_2 \to r_2\ [\varphi_2]$ with distinct variables (i.e., $\mathcal{V}ar(\ell_1, r_1, \varphi_1) \cap \mathcal{V}ar(\ell_2, r_2, \varphi_2) = \emptyset$) are all tuples $\langle s, t, \phi\rangle$ such that a non-variable subterm $\ell_1|_p$ of ℓ_1 at a position p is unifiable with ℓ_2, "$p \neq \varepsilon$, $\rho_1 \neq \rho_2$ up to variable renaming, or $\mathcal{V}ar(r_1) \subseteq \mathcal{V}ar(\ell_1)$", the most general unifier γ of $\ell_1|_p$ and ℓ_2 respects variables of both ρ_1 and ρ_2, i.e., $\gamma(x)$ is either a value or a variable for all variables x in $\mathcal{V}ar(\varphi_1, \varphi_2) \cup (\mathcal{V}ar(r_1, r_2) \setminus \mathcal{V}ar(\ell_1, \ell_2))$, $(\varphi_1 \wedge \varphi_2)\gamma$ is satisfiable, $s = r_1\gamma$, $t = (\ell_1[r_2]_p)\gamma$, and $\phi = (\varphi_1 \wedge \varphi_2)\gamma$. The set of critical pairs of an LCTRS $\mathcal{R}$ is denoted by $CP(\mathcal{R})$, which includes all critical pairs of two rules in $\mathcal{R} \cup \mathcal{R}_{calc}$. A critical pair $\langle s, t, \phi\rangle$ is called *trivial* if $s\ [\phi] \sim t\ [\phi]$. An LCTRS $\mathcal{R}$ is called *weakly orthogonal* if $\mathcal{R}$ is left-linear and all critical pairs of $\mathcal{R}$ are trivial.

Theorem 1 ([4]). *A weakly orthogonal LCTRS is confluent.*

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

Theorem 2 ([7]). *A terminating LCTRS is confluent if all its critical pairs are joinable.*

This version uses C03 [6][4]—a **CO**nverter for proving **CO**nfluence of **CO**nditional TRSs—to prove confluence of LCTRSs that can be considered *many-sorted TRSs* (MS-TRSs, for short): An LCTRS $\mathcal{R}$ is an MS-TRS if every rule $\ell \to r \Leftarrow \varphi$ satisfies all of the following:

- both ℓ and r are calculation-free,
- $\mathcal{V}ar(\ell) \supseteq \mathcal{V}ar(r)$, and
- φ is true.

For example, the LCTRS `1526.ari` in the ARI database can be considered an MS-TRS. Since we consider well-typed LCTRSs only, the MS-TRSs obtained from LCTRSs are well-typed. Since the techniques used in C03 do not rely on sorts, we transform an LCTRS considered an MS-TRS into a TRS by dropping sorts and guards.

[1] http://www.lctrs.jp/tools/craris/
[2] https://project-coco.uibk.ac.at/ARI/lctrs.php
[3] https://www.lctrs.jp/tools/crisys2/
[4] https://www.lctrs.jp/tools/co3/

References

[1] T. Aoto, N. Hirokawa, D. Kim, M. Kojima, A. Middeldorp, F. Mitterwallner, N. Nishida, T. Saito, J. Schöpf, K. Shintani, R. Thiemann, and A. Yamada. A new format for rewrite systems. In C. Chenavier and S. Winkler, editors, *Proceedings of the 12th International Workshop on Confluence*, pages 32–37, 2023.

[2] M. Kojima and N. Nishida. Reducing non-occurrence of specified runtime errors to all-path reachability problems of constrained rewriting. *Journal of Logical and Algebraic Methods in Programming*, 135:1–19, 2023.

[3] C. Kop. Termination of LCTRSs. In *Proceedings of the 13th International Workshop on Termination*, pages 1–5, 2013.

[4] C. Kop and N. Nishida. Term rewriting with logical constraints. In P. Fontaine, C. Ringeissen, and R. A. Schmidt, editors, *Proceedings of the 9th International Symposium on Frontiers of Combining Systems*, volume 8152 of *Lecture Notes in Artificial Intelligence*, pages 343–358. Springer, 2013.

[5] A. Matsumi, N. Nishida, M. Kojima, and D. Shin. On singleton self-loop removal for termination of LCTRSs with bit-vector arithmetic. In A. Yamada, editor, *Proceedings of the 19th International Workshop on Termination*, pages 1–6, 2023.

[6] N. Nishida and M. Kojima. CO3 (Version 2.6). In R. Gutiérrez and N. Nishida, editors, *Proceedings of the 14th International Workshop on Confluence*, pages 65–66, 2025.

[7] J. Schöpf and A. Middeldorp. Confluence criteria for logically constrained rewrite systems. In B. Pientka and C. Tinelli, editors, *Proceedings of the 29th International Conference on Automated Deduction*, volume 14132 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2023.

CoCo 2026 Participant: FORT-h 2.1

Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria
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 is a reimplementation of the tool FORT [4], but is based on a new variant of the decision procedure, described in [2], for the larger class of linear variable-separated rewrite systems. This variant supports a more expressive theory and is based on anchored ground tree transducers. More importantly, it can produce certificates for the YES/NO answers. These certificates can then be verified by FORTify, an independent Haskell program that is code-generated from the formalization of the decision procedure in the proof assistant Isabelle/HOL.

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

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

FORT-h participates in the following CoCo 2026 categories: COM, GCR, NFP, TRS, UNC, and UNR. In all six categories it additionally participates together with FORTify [3] to produce certified YES/NO answers.

FORT-h 2.1 accepts input problems in ARI[1] format.

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] Aart Middeldorp, Alexander Lochmann, and Fabian Mitterwallner. First-Order Theory of Rewriting for Linear Variable-Separated Rewrite Systems: Automation, Formalization, Certification. *Journal of Automated Reasoning*, 67(14):1–76, 2023. doi: 10.1007/s10817-023-09661-7.

[3] Aart Middeldorp. CoCo 2026 Participant: FORTify 2.1. In *Proc. 15th International Workshop on Confluence*, 2026. This volume.

[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.

[1] https://project-coco.uibk.ac.at/ARI/

CoCo 2026 Participant: FORTify 2.1

Aart Middeldorp

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

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

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

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

This year FORTify participates, together with FORT-h, in the following CoCo 2026 categories: COM, GCR, NFP, TRS, 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. Reducing Rewrite Properties to Properties on Ground Terms, 2022. https://isa-afp.org/entries/Rewrite_Properties_Reduction.html, Formal proof development.

[3] Alexander Lochmann and Bertram Felgenhauer. First-Order Theory of Rewriting. *Archive of Formal Proofs*, 2022. https://isa-afp.org/entries/FO_Theory_Rewriting.html, Formal proof development.

[4] Alexander Lochmann, Bertram Felgenhauer, Christian Sternagel, René Thiemann, and Thomas Sternagel. Regular Tree Relations. *Archive of Formal Proofs*, 2021. https://www.isa-afp.org/entries/Regular_Tree_Relations.html, Formal proof development.

[5] Aart Middeldorp, Alexander Lochmann, and Fabian Mitterwallner. First-Order Theory of Rewriting for Linear Variable-Separated Rewrite Systems: Automation, Formalization, Certification. *Journal of Automated Reasoning*, 67(14):1–76, 2023. doi: 10.1007/s10817-023-09661-7.

[6] Aart Middeldorp. CoCo 2026 Participant: FORT-h 2.1. In *Proc. 15th International Workshop on Confluence*, 2026. This volume.

AProVE'26: Confluence Analysis in a Termination Tool

J.-C. Kassing[12], A. Schüßler[13], T. Sokolowski[14], and J. Giesl[15]

[1] RWTH Aachen University, Aachen, Germany
[2] kassing@cs.rwth-aachen.de
[3] alwin.schuessler@rwth-aachen.de
[4] tobias.sokolowski@rwth-aachen.de
[5] giesl@informatik.rwth-aachen.de

AProVE (Automated Program Verification Environment) is a tool for fully automatic program verification. Its primary focus is on proving termination, analyzing (worst-case) complexity, and verifying safety or infeasibility for a variety of programming languages, including term rewriting. For further details on AProVE's general approach to these analyses, see [4].

Termination and confluence are two closely related properties. On the one hand, local confluence lets us reduce the termination analysis of overlay systems from arbitrary rewrite sequences to innermost ones, which has been shown to be substantially easier. On the other hand, confluence becomes decidable once termination is guaranteed. This interplay makes AProVE's powerful termination analysis a natural foundation for confluence analysis.

AProVE has participated in the annual Confluence Competition in 2025 for the first time. At first, it supported only a small set of techniques that had originally been implemented for the termination analysis of term rewrite systems. Since its initial participation, however, we have added more techniques to both prove and disprove confluence. This year, we mostly integrated techniques based on polynomial interpretations, since such interpretations already have been implemented in AProVE, due to their usage in termination analysis.

AProVE relies on the following techniques, where those labeled with ⋆ have been newly implemented since last year's competition:

- *Termination-based Confluence Analysis*: We use termination analysis to check whether the well-known decision procedure for confluence of terminating systems is applicable.
- *Modularity*: We exploit classical results on the modularity of confluence. More precisely, we implemented several of the modularity criteria presented in [3] and [6].
- ⋆ *Decreasing Diagrams Based on Polynomial Interpretations*: Polynomial interpretations can be used to automate the decreasing diagrams technique of [7] via [1].
- ⋆ *Utilization of Redundant Rewrite Rules*: Rules that can be simulated by other rules are called redundant, see [5]. Confluence analysis can benefit from both adding and removing such redundant rules.
- *Disproving via Dependency Graph Approximations*: To disprove joinability of critical pairs, AProVE reuses techniques originally designed for proving infeasibility in dependency graph approximations, e.g., checking unifiability between the target term and an approximation of the top part of the source term that is preserved during rewrite steps.
- ⋆ *Disproving Confluence via Polynomial Interpretations*: Polynomial interpretations can not only be used to prove confluence but also to disprove confluence. We mostly rely on the techniques described in [2].

References

[1] Takahito Aoto. Automated Confluence Proof by Decreasing Diagrams based on Rule-Labelling. In *Proc. RTA'10*, LIPIcs 6, pages 7–16, 2010.

[2] Takahito Aoto. Disproving Confluence of Term Rewriting Systems by Interpretation and Ordering. In *Proc. FroCoS'13*, LNCS 8152, pages 311–326, 2013.

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

[4] 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.

[5] Julian Nagele, Bertram Felgenhauer, and Aart Middeldorp. Improving Automatic Confluence Analysis of Rewrite Systems by Redundant Rules. In *Proc. RTA'15*, LIPIcs 36, pages 257–268, 2015.

[6] Enno Ohlebusch. *Advanced Topics in Term Rewriting.* Springer, New York, NY, 2002.

[7] Vincent van Oostrom. Confluence by Decreasing Diagrams. *Theoretical Computer Science*, 126:259–280, 1994.

CONFident at the 2026 Confluence Competition*

Raúl Gutiérrez and Salvador Lucas

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

1 Overview

CONFident is a tool which is able to prove confluence of TRSs, CS-TRSs, CTRSs and CS-CTRSs. Recently, CONFident has been extended to support generalized term rewriting systems (GTRSs), a class of term rewriting systems that integrates CS-CTRSs and Horn clauses [4, 5]. The tool is available here:

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

It is written in Haskell implementing the Confluence Framework. We implement the processors using the logical approach presented in [1, 3, 7] and mechanizing them by external tools like MU-TERM [3], infChecker [1, 6], AGES [2], Prover9 and Mace4 [9] and Barcelogic[1]. Last improvements of infChecker can be found in [8].

References

[1] R. Gutiérrez and S. Lucas. Automatically Proving and Disproving Feasibility Conditions. In N. Peltier and V. Sofronie-Stokkermans, editor, *Proc. of IJCAR'2020*, LNCS 12167:416–435. Springer, 2020.

[2] R. Gutiérrez and S. Lucas. Automatic Generation of Logical Models with AGES. In *CADE 2019: Automated Deduction - CADE 27*, LNCS 11716:287:299. Springer, 2019.

[3] R. Gutiérrez and S. Lucas. MU-TERM: Verify Termination Properties Automatically (System Description). In N. Peltier and V. Sofronie-Stokkermans, editor, *Proc. of IJCAR'2020*, LNCS 12167:436–447. Springer, 2020.

[4] R. Gutiérrez. and S. Lucas On Proving Confluence of Generalized Term Rewriting Systems Using CONFident. *Proc. of the 15th International Workshop on Confluence, IWC'26*, to appear, 2026.

[5] R. Gutiérrez. and S. Lucas Proving Confluence in the Confluence Framework with CONFident. *Fundamenta Informaticae*, 192, Issue 2: LOPSTR 2022, 2024.

[6] R. Gutiérrez and S. Lucas. Proving and disproving feasibility with infChecker. In *Proc. of the 14th International Workshop on Confluence, IWC'25*, 38, 2025.

[7] S. Lucas. Proving semantic properties as first-order satisfiability. *Artificial Intelligence* 277, paper 103174, 24 pages, 2019.

[8] S. Lucas. Confluence of Almost Parallel-Closed Generalized Term Rewriting Systems. In *Proc. of 30th International Conference on Automated Deduction, CADE-30*, LNAI 15943:187–206, 2025.

[9] W. McCune. Prover9 and Mace4. [online]. Available at https://www.cs.unm.edu/~mccune/mace4/.

*Partially supported by MCIN/AEI project PID2024-162030OB-100 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe" and by the grant CIPROM/2022/6 funded by Generalitat Valenciana.

[1] https://barcelogic.com/

infChecker at the 2026 Confluence Competition*

Raúl Gutiérrez and Salvador Lucas

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

1 Overview

infChecker is a tool for checking *(in)feasibility* of sequences of rewrite relations with respect to *first-order theories*, called goals [7]. Input systems can be TRSs, CS-TRSs, CTRSs, CS-CTRSs y GTRSs [6]. infChecker participates in the INF category at the Confluence Competition but it is also used as as a external tool in CONFident [1, 2], which participates in several categories in the Competition.

The tool is available here:

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

Some processors are mechanized using external tools like AGES [4], Prover9 and Mace4 [8]. Latest description of the tool can be found in [3, 5]. The set of relations accepted by infChecker are $\{\to, \to^*, \to^+, \hookrightarrow, \hookrightarrow^*, \hookrightarrow^+, \rhd, \trianglerighteq, \downarrow, \int, \leftrightarrow, \leftrightarrow\!\!\!\!\hookrightarrow, \leftrightarrow^*, \leftrightarrow\!\!\!\!\hookrightarrow^*\}$ [5].

References

[1] R. Gutiérrez. and S. Lucas On Proving Confluence of Generalized Term Rewriting Systems Using CONFident. *Proc. of the 15th International Workshop on Confluence, IWC'26*, to appear, 2026.

[2] R. Gutiérrez. and S. Lucas Proving Confluence in the Confluence Framework with CONFident. *Fundamenta Informaticae*, 192, Issue 2: LOPSTR 2022, 2024.

[3] 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.

[4] R. Gutiérrez and S. Lucas. Automatic Generation of Logical Models with AGES. In *CADE 2019: Automated Deduction - CADE 27*, LNCS 11716:287:299. Springer, 2019.

[5] R. Gutiérrez and S. Lucas. Proving and disproving feasibility with infChecker. In *Proc. of the 14th International Workshop on Confluence, IWC'25*, 38, 2025.

[6] S. Lucas. Local confluence of conditional and generalized term rewriting systems. *Journal of Logical and Algebraic Methods in Programming*, 136, paper 100926, pages 1-23, 2024.

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

[8] W. McCune. Prover9 and Mace4. [online]. Available at https://www.cs.unm.edu/~mccune/mace4/.

*Partially supported by MCIN/AEI project PID2024-162030OB-100 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe" and by the grant CIPROM/2022/6 funded by Generalitat Valenciana.

ACP: System Description for CoCo 2026

Takahito Aoto

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

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

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

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

References

[1] ACP (Automated Confluence Prover). http://www.nue.ie.niigata-u.ac.jp/tools/acp/.

[2] T. Aoto and Y. Toyama. Automated proofs of unique normal forms w.r.t. conversion for term rewriting systems. In *Proc. of 12th FroCoS*, volume 11715 of *LNAI*, pages 330–347. Springer-Verlag, 2019.

[3] T. Aoto, J. Yoshida, and Y. Toyama. Proving confluence of term rewriting system automatically. In *Proc. of 20th RTA*, volume 5595 of *LNCS*, pages 93–102. Springer-Verlag, 2009.

[4] T. Aoto. Proving uniqueness of normal forms w.r.t. reduction of term rewriting systems. In *Proc. of 34th LOPSTR*, volume 14919 of *LNCS*, pages 185–201. Springer-Verlag, 2024.

[5] M. Yamaguchi and T. Aoto, A fast decision procedure for uniqueness of normal forms w.r.t. conversion of shallow term rewriting systems. In *Proc. of 5th FSCD*, volume 167 of *LIPIcs*, pages 9:1–9:23. Schloss Dagstuhl, 2020.

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

AGCP: System Description for CoCo 2026

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. In this year, no new ground (non-)confluence criterion has been incorporated from the one submitted for CoCo 2023. Below we provide a brief overview of AGCP.

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 $u\theta_g \overset{*}{\underset{\succsim}{\longleftrightarrow}}_{\mathcal{R}} v\theta_g$ for any its ground instance $u\sigma_g \approx v\sigma_g$, where $x \overset{*}{\underset{\succsim}{\longleftrightarrow}} y$ 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].

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.

CoCo 2026 Participant: nonreach + CeTA-Prover

Florian Meßner[1] and René Thiemann[2]

[1] Independent Researcher, Austria
[2] University of Innsbruck, Austria

The tool nonreach [1,2] is an automated, efficient tool to check infeasibility with respect to oriented conditional term rewrite systems (CTRSs). The Haskell source code can be obtained from a public *git* repository hosted on *codeberg*, and during the installation process it will automatically download CeTA-Prover [3].

https://codeberg.org/fmessner/nonreach

Given a CTRS (or a TRS) and one or more infeasibility problems, nonreach uses a combination of *decomposition*, based on narrowing (with some heuristics) and proving root-nonreachability [4], and *fast checks*, based on *tcap* and the *inductive symbol transition graph* [4].

These methods are applied alternately until nonreach either obtains infeasibility (by simplifying the tree to False), finds a satisfying substitution, or reaches a user-defined threshold of iterations (and concludes MAYBE).

Whereas the proofs of infeasibility of the previous version of nonreach are quite verbose and can also be machine checked, this was not the case for feasibility. In particular, providing just the satisfying substitution σ such that $s\sigma \to^* t\sigma$ for some reachability condition $s \approx t$ is not enough information for the CPF certificates that are expected by CeTA [5].

To this end the new version of nonreach, version 1.4, now invokes CeTA-Prover internally to reconstruct all the details of the rewrite sequence $s\sigma \to^* t\sigma$. These details are then displayed in human-readable form (by nonreach) or they are used to generate machine checkable certificates (by CeTA-Prover).

In total, our new combination of nonreach + CeTA-Prover is the first automatic tool that generates *certifiable feasibility proofs*. Moreover, there are minor extensions on the nonreach side, e.g., by adding native support for the ARI format and for CPF 3 certificates, i.e., without any external translation tools from COPS or from CPF 2.

References

[1] Florian Meßner and Christian Sternagel. nonreach – A Tool for Nonreachability Analysis. *Proc. of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 11427 of *LNCS*, pages 337–343. Spring, 2019. URL: doi:10.1007/978-3-030-17462-0_19.

[2] Florian Meßner. Automated Conditional Nonreachability Master Thesis, University of Innsbruck, 2020. https://diglib.uibk.ac.at/ulbtirolhs/content/titleinfo/5489915.

[3] René Thiemann. CoCo 2026 Participant: CeTA-Prover. *Proc. of the 15th International Workshop on Confluence, July 24, 2026*. This volume.

[4] Christian Sternagel and Akihisa Yamada. Reachability analysis for termination and confluence of rewriting. *Proc. of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 11427 of *LNCS*, pages 262–278. Springer, 2019. doi:10.1007/978-3-030-17462-0_15.

[5] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. *Proc. of the International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009. doi:10.1007/978-3-642-03359-9_31.