# IWC 2013

## 2nd International Workshop on Confluence

Proceedings

Editors: Nao Hirokawa & Vincent van Oostrom

June 28, 2013, Eindhoven, The Netherlands

# Preface

This report contains the proceedings of the *2nd International Workshop on Confluence (IWC 2013)*. The workshop was held in Eindhoven on June 28, 2013 as part of the 7th International Conference on Rewriting, Deduction, and Programming (RDP 2013). The 1st IWC took place in Nagoya (2012).

Recently there is a renewed interest in confluence research, resulting in new techniques, tool support as well as new applications. The workshop promotes and stimulates research and collaboration on confluence and related properties. It encourages the presentation of new directions, developments, and results as well as surveys and tutorials on existing knowledge in this area. In addition to original contributions, the workshop solicited short versions of recently published articles and papers submitted elsewhere.

IWC 2013 received 10 submissions. Each submission was reviewed by 3 program committee members. After deliberations the program committee decided to accept all submissions, which are contained in this report. Apart from these contributed talks, the workshop had an invited talk by *Patrick Dehornoy* on *Three Termination Problems*, and jointly with the 1st Workshop on Infinitary Rewriting (WIR 2013), an invited talk by *Jan Willem Klop* on *Confluence and Infinity - a kaleidoscopic view*. Their abstracts are also included in the report. Moreover, the 2nd Confluence Competition (CoCo 2013) was held during the workshop and the results are available at `http://coco.nue.riec.tohoku.ac.jp/2013/`.

Several persons helped to make IWC 2013 a success. We are grateful to the members of the program committee for their work. Special thanks go to Harald Zankl for organizing CoCo. Last but not least, we thank Hans Zantema for his indispensable help.

*Kanazawa & Utrecht, June 2013* *Nao Hirokawa & Vincent van Oostrom*

# Program Committee

| | | |
|---|---|---|
| Guillem Godoy | *Technical University of Catalonia* | |
| Nao Hirokawa | *JAIST* | (co-chair) |
| Barbara König | *University of Duisburg-Essen* | |
| Vincent van Oostrom | *Utrecht University* | (co-chair) |
| Michio Oyamaguchi | *Nagoya University* | |
| Harald Zankl | *University of Innsbruck* | |
| Hans Zantema | *Eindhoven University of Technology* | |

# Table of Contents

## Abstracts of Invited Talks



## Contributed Papers

# Three Termination Problems

Patrick Dehornoy

University of Caen

We shall describe three termination problems originating from various areas of algebra. These are the termination of handle reduction in braid theory, where termination is proved but with a very coarse complexity bound, the termination of the Polish algorithm in the theory of self-distributive systems, where termination is not yet proved, and the termination of subword reversing in semigroup theory, where termination is proved in some cases. In all three situations, the results known so far rely on the specific properties of the underlying objects, and it would be highly desirable to know whether general techniques might help.

N. Hirokawa & V. van Oostrom (eds.); 2nd International Workshop on Confluence, pp. 1–1

# Confluence and Infinity - a kaleidoscopic view

Jan Willem Klop

VU University Amsterdam and CWI Amsterdam

In this talk we will attempt to present some highlights in old and new studies concerning the pivotal notion of confluence in lambda calculus, term rewriting and infinitary rewriting. We first aim the kaleidoscope on the classics: lambda calculus (Tait-Martin-Löf, Aczel, superdevelopments, Parallel Moves Lemma), combinatory logic and other orthogonal systems, deviating from that ideal setting by looking briefly at non-left-linear rules, such as Surjective Pairing (De Vrijer), confluence for braids, and confluence by completion (Zantema, van Oostrom)

In a second part, we rotate the kaleidoscope and view some basic confluence methods, Newman's Lemma, De Bruijn's Weak Diamond Property, Van Oostrom's Decreasing Diagrams.

In a third twist of the kaleidoscope we invoke the setting of infinitary rewriting, for orthogonal rewriting, and also for lambda calculus. We briefly show a recent remarkable coinductive setup of infinitary rewriting, and then return to infinitary confluence, signalling some striking differences with the finitary rewriting world, with as motto: confluence lost, confluence regained. We describe the Threefold Path to regain confluence, via the semantics of Böhm Trees and its two variants. Speaking about Böhm trees, we (might) turn to a more refined notion, clocked Böhm Trees, a recent method to discriminate lambda terms with respect to finitary convertibility, very suitable for fixed point combinators.

Somewhere in between we (might) show that the eta-rule wreaks havoc in infinitary lambda calculus. We conclude with the miracle of infinitary confluence for lambda-beta-Omega calculus.

# Disproving Confluence of Term Rewriting Systems by Interpretation and Ordering (extended abstract)

Takahito Aoto

RIEC, Tohoku University
2-1-1 Katahira, Aoba-ku, Sendai, 980-8577, Japan
aoto@nue.riec.tohoku.ac.jp

**Abstract**

We present new criteria for ensuring non-joinability of terms based on interpretation and ordering, and report on an implementation of confluence disproving procedure based on some instances of the criteria. The experiment reveals that our methods can be applied to automatically disprove confluence of some term rewriting systems, on which state-of-the-art automated confluence provers fail.

## 1 Introduction

In contrast to many dedicated techniques that have been developed to prove confluence of term rewriting systems, not many techniques for disproving confluence are known. A typical approach to disprove confluence of (non-terminating) TRSs is first to construct some candidates of two terms that can be reduced from a common term, and then to show that these candidates are not joinable, i.e. they do not have a common reduct. In this scenario, as well as the selection of the candidates, proving non-joinability of terms is essential. So far, the only serious approach to prove the non-joinability of terms is to use approximation by tree automata [4, 7].

In this paper, we give new methods for proving that given two terms $s, t$ are not joinable. The first method consists in giving an interpretation, e.g. a mapping from terms to natural numbers, that is preserved by the application of usable rules and such that the interpretation of $s$ is different from that of $t$. The second method consists in giving an ordering $>$ such that $s > t$, and usable rules from $s$ only increase or preserve w.r.t. $>$ and the usable rules from $t$ only decrease or preserve w.r.t. $>$. These methods are implemented using polynomial interpretations and recursive path orderings—interpretations and orderings that are widely used in the literature for termination proving. The experiment reveals that our methods can be applied to automatically disprove confluence of some term rewriting systems, on which state-of-the-art automated confluence provers fail to disprove.

## 2 Preliminaries

We assume familiarity with standard notions and notations on term rewriting (see e.g. [3]). Below we explain some extra notations used in the paper. The *disjoint union* of two sets $A$ and $B$ is denoted by $A \uplus B$, and that of all $A_i$ $(i \in I)$ by $\biguplus_{i \in I} A_i$. The set of terms over a set $\mathcal{F}$ of function symbols and the set $\mathcal{V}$ of variables is denoted by $\mathrm{T}(\mathcal{F}, \mathcal{V})$. The set of variables in a term $t$ is denoted by $\mathcal{V}(t)$. We write $s \trianglelefteq t$ to denote that $s$ is a subterm of $t$. We write $\mathrm{Unif}(s, t)$ to denote that the terms $s$ and $t$ are unifiable. A *rewrite rule* $l \to r$ is a pair of terms; we here drop the usual restriction that $l \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. A *rewrite relation* is a

relation on terms that is closed under contexts and substitutions. A strict partial order (partial order, quasi-order) is a *rewrite strict partial order* (*rewrite partial order*, *rewrite quasi-order*, respectively) if it is a rewrite relation.

Given a term $s$, the sets of terms $\{t \in \mathrm{T}(\mathcal{F},\mathcal{V}) \mid s \xrightarrow{*} t\}$ and $\{t \in \mathrm{T}(\mathcal{F},\mathcal{V}) \mid t \xrightarrow{*} s\}$ are denoted by $[s](\xrightarrow{*})$ and $(\xrightarrow{*})[s]$, respectively. Terms $s$ and $t$ are said to be *joinable* if $[s](\xrightarrow{*}) \cap [t](\xrightarrow{*}) \neq \emptyset$, and *non-joinable* otherwise. We write $\mathrm{NJ}(s,t)$ to denote that the terms $s$ and $t$ are non-joinable. In order to disprove that a TRS $\mathcal{R}$ is confluent, we construct two terms $s$ and $t$ such that $(\xrightarrow{*})[s] \cap (\xrightarrow{*})[t] \neq \emptyset$ in some way, and then prove $\mathrm{NJ}(s,t)$. From here on, we concentrate on the problem of proving $\mathrm{NJ}(s,t)$, the non-joinability problem.

# 3 Proving Non-Joinability by Interpretation

In this section, we present several criteria to prove non-joinability of terms based on their interpretations in $\mathcal{F}$-algebras.

An *$\mathcal{F}$-algebra* $\mathcal{A} = \langle A, \langle f^{\mathcal{A}} \rangle_{f\in\mathcal{F}} \rangle$ is a pair of a set $A$ and a tuple of functions $f^{\mathcal{A}} : A^n \to A$ for each $n$-ary function symbol $f \in \mathcal{F}$. The set $A$ is called the *carrier set* of the $\mathcal{F}$-algebra $\mathcal{A}$ and is denoted by $|\mathcal{A}|$. A *valuation* on the $\mathcal{F}$-algebra $\mathcal{A}$ is a mapping $\mathcal{V} \to A$. Suppose an $\mathcal{F}$-algebra $\mathcal{A} = \langle A, (f^{\mathcal{A}})_{f\in\mathcal{F}} \rangle$ is fixed. Then the *interpretation* of a term under the valuation $\sigma$ is denoted by $[\![t]\!]_\sigma$.

The notion of usable rules [2] is well-known in the literature for proving termination of TRSs. We introduce a notion of usable rules for non-joinability suitable for our setting. For this, the notion of TCAP [5] is used. For terms $t$, $\mathrm{TCAP}(t)$ is defined recursively like this: $\mathrm{TCAP}(x) = x'$, $\mathrm{TCAP}(f(t_1,\ldots,t_n)) = x'$ if $\mathrm{Unif}(f(u_1,\ldots,u_n), l)$ for some $l \to r \in \mathcal{R}$, and $\mathrm{TCAP}(f(t_1,\ldots,t_n)) = f(u_1,\ldots,u_n)$ otherwise, where $u_i = \mathrm{TCAP}(t_i)$ $(1 \le i \le \mathrm{arity}(f))$. Here, a new fresh variable is taken for $x'$ every time it is used. Our notion of usable rules is obtained from the one for innermost termination [5] by replacing ICAP with TCAP.

**Definition 1** (usable rules)**.** *The set of* usable rules *for non-joinability w.r.t. TRS $\mathcal{R}$ and a term $s$ is the smallest set $\mathcal{U}_{\mathsf{nj}}(\mathcal{R},s) \subseteq \mathcal{R}$ satisfying two conditions: (i) for any $l \to r \in \mathcal{R}$ and non-variable subterm $f(u_1,\ldots,u_n) \trianglelefteq s$, if $\mathrm{Unif}(f(\mathrm{TCAP}(u_1),\ldots,\mathrm{TCAP}(u_n)), l)$ then $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R},s)$; (ii) if $l' \to r' \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R},s)$ and $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R},r')$, then $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R},s)$.*

The following is a key lemma for proving our theorem given below.

**Lemma 2.** *Let $\mathcal{R}$ be a TRS, $l \to r \in \mathcal{R}$ and $s,t$ terms. If $s \xrightarrow{*}_{\mathcal{R}} \circ \to_{\{l\to r\}} t$ then $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R},s)$.*

**Theorem 3.** *Let $s,t$ be terms and $\mathcal{A} = \langle A, \langle f^{\mathcal{A}} \rangle_{f\in\mathcal{F}} \rangle$ an $\mathcal{F}$-algebra such that $A = \biguplus_{i\in I} A_i$. Suppose (i) for any valuation $\sigma$ and $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R},s) \cup \mathcal{U}_{\mathsf{nj}}(\mathcal{R},t)$, if $[\![l]\!]_\sigma \in A_i$ then $[\![r]\!]_\sigma \in A_i$, (ii) for any $f \in \mathcal{F}$, $a \in A$ and $i,j \in I$, if $a \in A_i$ implies $f^{\mathcal{A}}(\ldots,a,\ldots) \in A_j$, then $f^{\mathcal{A}}(\ldots,b,\ldots) \in A_j$ for any $b \in A_i$ and (iii) $[\![s]\!]_\rho \in A_i$ and $[\![t]\!]_\rho \in A_j$ for some valuation $\rho$ and $i \neq j$. Then $\mathrm{NJ}(s,t)$.*

The criterion of Theorem 3, in general, is not amenable for automation, and one has to use more concrete instances of the theorem such as given below.

**Corollary 4.** *Let $\mathcal{A}$ be an $\mathcal{F}$-algebra and $s,t$ be terms. Suppose (i) $[\![l]\!]_\sigma = [\![r]\!]_\sigma$ for any valuation $\sigma$ and $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R},s) \cup \mathcal{U}_{\mathsf{nj}}(\mathcal{R},t)$ and (ii) $[\![s]\!]_\rho \neq [\![t]\!]_\rho$ for some valuation $\rho$. Then $\mathrm{NJ}(s,t)$.*

**Corollary 5.** *Let $s, t$ be terms and $\mathcal{A}$ an $\mathcal{F}$-algebra whose carrier set is a set of integers. Suppose there exists an integer $k \geq 2$ such that (i) for any valuation $\sigma$ and $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, s) \cup \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, t)$, $[\![l]\!]_\sigma \equiv [\![r]\!]_\sigma \pmod{k}$ and (ii) $[\![s]\!]_\rho \not\equiv [\![t]\!]_\rho \pmod{k}$ for some valuation $\rho$. Then* $\mathrm{NJ}(s, t)$.

In following examples, non-confluence is shown using these corollaries.

**Example 6.** *Let $\mathcal{R} = \{(1) : \mathsf{a} \to \mathsf{h}(\mathsf{c}), (2) : \mathsf{a} \to \mathsf{h}(\mathsf{f}(\mathsf{c})), (3) : \mathsf{h}(x) \to \mathsf{h}(\mathsf{h}(x)), (4) : \mathsf{f}(x) \to \mathsf{f}(\mathsf{g}(x))\}$. Let $s = \mathsf{h}(\mathsf{c})$ and $t = \mathsf{h}(\mathsf{f}(\mathsf{c}))$. As $\mathsf{a} \in (\xrightarrow{*})[s] \cap (\xrightarrow{*})[t]$, it suffices to show* $\mathrm{NJ}(s, t)$ *to disprove the confluence of $\mathcal{R}$. We have $\mathcal{U}_{\mathsf{nj}}(\mathcal{R}, s) \cup \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, t) = \{(3), (4)\}$. Take an $\mathcal{F}$-algebra $\mathcal{A} = \langle \{0, 1\}, \langle f^{\mathcal{A}} \rangle_{f \in \mathcal{F}} \rangle$ as $\mathsf{a}^{\mathcal{A}} = \mathsf{c}^{\mathcal{A}} = 0$, $\mathsf{f}^{\mathcal{A}}(n) = 1 - n$, $\mathsf{h}^{\mathcal{A}}(n) = \mathsf{g}^{\mathcal{A}}(n) = n$. Then for any valuation $\sigma$, we have $[\![\mathsf{h}(x)]\!]_\sigma = \sigma(x) = [\![\mathsf{h}(\mathsf{h}(x))]\!]_\sigma$ and $[\![\mathsf{f}(x)]\!]_\sigma = 1 - \sigma(x) = [\![\mathsf{f}(\mathsf{g}(x))]\!]_\sigma$; thus, $[\![l]\!]_\sigma = [\![r]\!]_\sigma$ for each $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, s) \cup \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, t)$. Take an arbitrary valuation $\rho$. Then $[\![s]\!]_\rho = [\![\mathsf{h}(\mathsf{c})]\!]_\rho = 0 \neq 1 = [\![t]\!]_\rho = [\![\mathsf{h}(\mathsf{f}(\mathsf{c}))]\!]_\rho$. Therefore,* $\mathrm{NJ}(s, t)$ *by Corollary 4.*

**Example 7.** *Let $\mathcal{R} = \{(1) : \mathsf{a} \to \mathsf{f}(\mathsf{c}), (2) : \mathsf{a} \to \mathsf{h}(\mathsf{c}), (3) : \mathsf{f}(x) \to \mathsf{h}(\mathsf{g}(x)), (4) : \mathsf{h}(x) \to \mathsf{f}(\mathsf{g}(x))\}$. Let $s = \mathsf{f}(\mathsf{c})$ and $t = \mathsf{h}(\mathsf{c})$. We have $\mathcal{U}_{\mathsf{nj}}(\mathcal{R}, s) \cup \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, t) = \{(3), (4)\}$. Take an $\mathcal{F}$-algebra $\mathcal{A} = \langle \mathbb{N}, \langle f^{\mathcal{A}} \rangle_{f \in \mathcal{F}} \rangle$ as $\mathsf{a}^{\mathcal{A}} = \mathsf{c}^{\mathcal{A}} = 0$, $\mathsf{g}^{\mathcal{A}}(n) = n + 1$, $\mathsf{f}^{\mathcal{A}}(n) = n$, $\mathsf{h}^{\mathcal{A}}(n) = n + 1$. Then $[\![\mathsf{f}(x)]\!]_\sigma - [\![\mathsf{h}(\mathsf{g}(x))]\!]_\sigma = \sigma(x) - (\sigma(x) + 2) = -2$ and $[\![\mathsf{h}(x)]\!]_\sigma - [\![\mathsf{f}(\mathsf{g}(x))]\!]_\sigma = (\sigma(x) + 1) - (\sigma(x) + 1) = 0$. Take $k = 2$. Then $[\![\mathsf{f}(x)]\!]_\sigma \equiv [\![\mathsf{h}(\mathsf{g}(x))]\!]_\sigma \pmod{k}$ and $[\![\mathsf{h}(x)]\!]_\sigma \equiv [\![\mathsf{f}(\mathsf{g}(x))]\!]_\sigma \pmod{k}$ for any valuation $\sigma$. Furthermore, since we have $[\![s]\!]_\rho = [\![\mathsf{f}(\mathsf{c})]\!]_\rho = 0$ and $[\![t]\!]_\rho = [\![\mathsf{h}(\mathsf{c})]\!]_\rho = 1$, $[\![s]\!]_\rho \not\equiv [\![t]\!]_\rho \pmod{k}$. Hence,* $\mathrm{NJ}(s, t)$ *by Corollary 5.*

# 4 Proving Non-Joinability by Ordering

In Corollary 5, we considered the case that the carrier set is a set of integers. In such a case, another obvious choice to obtain a partition of the carrier set is to divide it as $A = \{n \in A \mid n < k\} \uplus \{n \in A \mid k \leq n\}$ for some $k$. We first formulate this idea in a more abstract setting, using the notion of ordered $\mathcal{F}$-algebra [10].

An *ordered $\mathcal{F}$-algebra* $\mathcal{A} = \langle A, \leq, \langle f^{\mathcal{A}} \rangle_{f \in \mathcal{F}} \rangle$ is a triple of a set $A$, a partial order $\leq$ on it and a tuple of functions $f^{\mathcal{A}} : A^n \to A$ for each $n$-ary function symbol $f \in \mathcal{F}$. We use $<$ to denote strict part of $\leq$, i.e. $< = \leq \setminus \geq$. An ordered $\mathcal{F}$-algebra $\mathcal{A} = \langle A, \leq, \langle f^{\mathcal{A}} \rangle_{f \in \mathcal{F}} \rangle$ is said to be *weakly monotone* if $a \leq b$ implies $f^{\mathcal{A}}(\ldots, a, \ldots) \leq f^{\mathcal{A}}(\ldots, b, \ldots)$ for any $a, b \in A$ and $f \in \mathcal{F}$.

**Theorem 8.** *Let $\mathcal{A}$ be a weakly monotone ordered $\mathcal{F}$-algebra and $s, t$ be terms. Suppose (i) $[\![l]\!]_\sigma \leq [\![r]\!]_\sigma$ for any valuation $\sigma$ and any $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, s)$, (ii) $[\![l]\!]_\sigma \geq [\![r]\!]_\sigma$ for any valuation $\sigma$ and any $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, t)$ and (iii) $[\![s]\!]_\rho > [\![t]\!]_\rho$ for some valuation $\rho$. Then* $\mathrm{NJ}(s, t)$.

We next consider the case that term algebras are taken as $\mathcal{F}$-algebras, and formulate the theorem in a more general way using the notion of rewrite relation. For this, the following notion is useful.

**Definition 9** (discrimination pair)**.** *A pair $\langle \gtrsim, \succ \rangle$ of two relations $\gtrsim$ and $\succ$ is said to be a* discrimination pair *if (i) $\gtrsim$ is a rewrite relation, (ii) $\succ$ is a strict partial order and (iii) $\gtrsim \circ \succ \subseteq \succ$ and $\succ \circ \gtrsim \subseteq \succ$.*

Clearly, for any rewrite quasi-order $\gtrsim$, the pair $\langle \gtrsim, \gtrsim \setminus \lesssim \rangle$ forms a discrimination pair.

Before presenting the next theorem, another notion from termination proving is required. An *argument filtering* [2] is a mapping such that $\pi(f) \in \{[i_1, \ldots, i_k] \mid i_1 < \cdots < i_k, 1 \leq i_1, \ldots, i_k \leq \mathrm{arity}(f)\} \cup \{i \mid 1 \leq i \leq \mathrm{arity}(f)\}$. Then the application $t^\pi$ of the argument filtering $\pi$ to terms $t$ is given by $x^\pi = x$ for $x \in \mathcal{V}$, $f(t_1, \ldots, t_n)^\pi = f(t_{i_1}^\pi, \ldots, t_{i_k}^\pi)$ if $\pi(f) = [i_1, \ldots, i_k]$, $f(t_1, \ldots, t_n)^\pi = t_i^\pi$ if $\pi(f) = i$. For a TRS $\mathcal{R}$, we put $\mathcal{R}^\pi = \{l^\pi \to r^\pi \mid l \to r \in \mathcal{R}\}$.

**Theorem 10.** *Let $\mathcal{R}$ be a TRS and $s, t$ terms. Suppose there exist a discrimination pair $\langle \gtrsim, \succ \rangle$ and an argument filtering $\pi$ such that $\mathcal{U}_{\mathsf{nj}}(\mathcal{R}^\pi, s^\pi) \subseteq \lesssim$, $\mathcal{U}_{\mathsf{nj}}(\mathcal{R}^\pi, t^\pi) \subseteq \gtrsim$ and $s^\pi \succ t^\pi$. Then* $\mathrm{NJ}(s, t)$.

In terms of interpretations, Theorem 10 amounts to take term algebras as $\mathcal{F}$-algebras, while Theorem 8 allows to take any $\mathcal{F}$-algebra. On the other hand, in terms of discrimination pairs, Theorem 8 amounts to take a discrimination pair of the form $\langle \gtrsim, \gtrsim \setminus \lesssim \rangle$.

**Example 11.** *Let $\mathcal{R} = \{(1) : \mathsf{c} \to \mathsf{f}(\mathsf{c}, \mathsf{d}), (2) : \mathsf{c} \to \mathsf{h}(\mathsf{c}, \mathsf{d}), (3) : \mathsf{f}(x, y) \to \mathsf{h}(\mathsf{g}(y), x), (4) : \mathsf{h}(x, y) \to \mathsf{f}(\mathsf{g}(y), x)\}$. Let $s = \mathsf{h}(\mathsf{f}(\mathsf{c}, \mathsf{d}), \mathsf{d})$ and $t = \mathsf{f}(\mathsf{c}, \mathsf{d})$. Take an argument filtering $\pi$ as $\pi(\mathsf{g}) = 1$, $\pi(\mathsf{f}) = [2]$ and $\pi(\mathsf{h}) = [1]$. Then we have $\mathcal{U}_{\mathsf{nj}}(\mathcal{R}^\pi, s^\pi) = \mathcal{U}_{\mathsf{nj}}(\mathcal{R}^\pi, t^\pi) = \{(3)^\pi, (4)^\pi\}$. The constraint $\{\mathsf{h}(\mathsf{f}(\mathsf{d})) \succ \mathsf{f}(\mathsf{d}), \mathsf{f}(y) \simeq \mathsf{h}(y), \mathsf{h}(x) \simeq \mathsf{f}(x)\}$ is satisfied by a discrimination pair $\langle \gtrsim_{rpo}, \gtrsim_{rpo} \setminus \lesssim_{rpo} \rangle$, where $\gtrsim_{rpo}$ is the recursive path order based on the precedence $\mathsf{f} \simeq \mathsf{h}$. Thus* $\mathrm{NJ}(s, t)$ *by Theorem 10.*

# 5 Implementations and Experiments

**Implementations** The following instances of presented criteria have been implemented. We assume below that we check non-joinability of ground terms $s, t$.

**Cor. 5 ($k = 2, 3$)** Corollary 5 applied for the polynomial interpretation with linear polynomials. In case $k = 2$, we check whether $[\![l]\!]_\sigma - [\![r]\!]_\sigma$ is even for all rewrite rules $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, s) \cup \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, t)$ and whether $[\![s]\!] - [\![t]\!]$ is odd. We encode these constraints in boolean formulas and check the constraints by an external SAT solver. We deal with integer variables of the range between 0 and 15. The case $k = 3$ is similar.

**Th. 8 (poly)** Theorem 8 applied for polynomial interpretation with linear polynomials. Similar to the case **Cor. 5 ($k = 2, 3$)**, we encode the constraints in boolean formulas and check the constraints by an external SAT solver. Our implementation tries two possible applications of the Theorem to show $\mathrm{NJ}(s, t)$, namely that (1) $[\![s]\!] > [\![t]\!]$, $[\![l]\!]_\sigma \geq [\![r]\!]_\sigma$ for $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, t)$ and $[\![l]\!]_\sigma \leq [\![r]\!]_\sigma$ for $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, s)$, and (2) $[\![t]\!] > [\![s]\!]$, $[\![l]\!]_\sigma \geq [\![r]\!]_\sigma$ for $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, s)$ and $[\![l]\!]_\sigma \leq [\![r]\!]_\sigma$ for $l \to r \in \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, t)$.

**Th. 10 (rpo)** Theorem 10 applied for recursive path order with argument filtering. Similar to the cases **Cor. 5 ($k = 2, 3$)** and **Th. 8 (poly)**, we encode the constraints in boolean formulas and check the constraints by an external SAT solver. We approximate the set of usable rules $\mathcal{U}_{\mathsf{nj}}(\mathcal{R}^\pi, s^\pi)$ first by $\mathcal{S} = \mathcal{U}_{\mathsf{nj}}(\mathcal{R}, s)$ before encoding and then $\mathcal{U}(\mathcal{S}^\pi, s^\pi)$, the set of usable rules for dependency pairs [2], at the time of encoding (and similarly for $\mathcal{U}_{\mathsf{nj}}(\mathcal{R}^\pi, t^\pi)$).

Candidates for the non-joinability test are generated from the input TRS $\mathcal{R}$ like this: (1) first compute the one-step unfolding $\mathcal{R}'$ of $\mathcal{R}$ [8] and then (2) compute critical pairs of $\mathcal{R} \cup \mathcal{R}'$, and finally, (3) all critical pairs are sorted w.r.t. term size and at most 100 crucial pairs are considered for candidates for non-joinability test.

**Experiments** Experiments have been performed on our implementation and the state-of-the-art confluence provers ACP [1] (ver. 0.31), CSI [9] (ver. 0.2) and Saigawa [6] (ver. 1.4). Each test is performed on a PC with one 2.50GHz CPU and 4G memory; the timeout is set to 60 seconds. We have tested a collection of 23 new examples which includes Examples 6, 7, 11 and their

Table 1: Summary of experiments

| | ACP | CSI | Saigawa | Cor. 5 ($k=2$) | Cor. 5 ($k=3$) | Th. 8 (poly) | Th. 10 (rpo) | all |
|---|---|---|---|---|---|---|---|---|
| Example 6 | × | × | × | ✓ | ✓ | ✓ | ✓ | ✓ |
| Example 7 | × | × | × | ✓ | ✓ | × | × | ✓ |
| Example 11 | × | × | × | × | × | × | ✓ | ✓ |
| 23 examples (success) | 9 | 12 | 3 | 16 | 16 | 14 | 19 | 21 |
| 23 examples (time in sec.) | 2 | 2107 | 228 | 25 | 293 | 206 | 26 | 84 |
| 35 examples (success) | 18 | 21 | 17 | 17 | 16 | 17 | 17 | 16 |
| 35 examples (time in sec.) | 71 | 485 | 482 | 318 | 562 | 446 | 106 | 761 |

variants, and a collection of 35 examples from the 1st Confluence Competition (CoCo 2012) that were not proved to be confluent by any of participating provers.

A summary of the experiments is shown in Table 1. The column below **all** denotes the result for the combination of the four instances. All provers ACP, CSI and Saigawa fail on Examples 6, 7 and 11. For the collection of 23 new examples, the following are observed: **Cor. 5** ($k = 2$) and **Cor. 5** ($k = 3$) succeed at the same examples. Examples handled by **Th. 8 (poly)** are also handled by **Th. 10 (rpo)** and also by **Cor. 5**. Examples handled by any of the provers ACP, CSI and Saigawa also are handled by **all**. For the collection of 35 examples from CoCo 2012, the following are observed: All instances succeed on the same examples, except for **Cor. 5** ($k = 3$), in which one timeouts. The numbers of examples on which ACP, CSI and Saigawa succeed but **all** fails are 4, 5, 3, respectively. Finally, the running time is observed like this: **Th. 10 (rpo)** $<$ **Cor. 5** ($k = 2$) $\ll$ **Th. 8 (poly)** $\ll$ **Cor. 5** ($k = 3$). All details of the experiments are available on the webpage: `http://www.nue.riec.tohoku.ac.jp/tools/acp/experiments/iwc13/all.html`.

# References


[1] T. Aoto, Y. Yoshida, and Y. Toyama. Proving confluence of term rewriting systems automatically. In *Proc. of 20th RTA*, volume 5595 of *LNCS*, pages 93–102. Springer-Verlag, 2009.

[2] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1–2):133–178, 2000.

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

[4] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proc. of 9th RTA*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.

[5] J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proc. of 5th FroCoS*, volume 3717, pages 216–231. Springer-Verlag, 2005.

[6] N. Hirokawa and D. Klein. Saigawa: A confluence tool. In *Proc. of 1st IWC*, page 49, 2012.

[7] A. Middeldorp. Approximating dependency graphs using tree automata techniques. In *Proc. of the 1st IJCAR*, volume 2083 of *LNAI*, pages 593–610. Springer-Verlag, 2001.

[8] É. Payet. Loop detection in term rewriting using eliminating unfoldings. *Theoretical Computer Science*, 403:307–327, 2008.

[9] H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – A confluence tool. In *Proc. of 23rd CADE*, volume 6803 of *LNAI*, pages 499–505. Springer-Verlag, 2011.

[10] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.

# Automatically Finding Non-confluent Examples in Term Rewriting

Hans Zantema
University of Technology, Eindhoven, The Netherlands
Radboud University, Nijmegen, The Netherlands
h.zantema@tue.nl

**Abstract**

Last year we presented a technique based on SAT solving to find counter examples in finite abstract rewriting fully automatically, in particular satisfying non-confluence. This note extends this work to term rewriting.

## 1 Introduction

It is well-known that local confluence does not imply confluence; the simplest example is the rewriting system on four constants $a, b, c, d$ having the four rules $a \to b, b \to a, a \to c, b \to d$.

In our earlier paper [7] the main goal was to find such examples fully automatically, that is, given a combination of properties like termination, confluence and several variants, an example of a rewriting system is found over finitely many constants satisfying the given properties. We described our tool `Carpa` together with its input language for giving such a combination of properties, that automatically generates such examples via a SAT solver. Indeed the above example is found automatically by entering the combination of local confluence and the negation of confluence.

However, `Carpa` looks for rewriting systems only over constants, that is, for finite abstract reduction systems (ARSs). For some combinations of properties such a finite ARS does not exist, while in a richer class of rewriting systems examples are found easily. As a simple example of this phenomenon assume we are looking for a rewriting system that is terminating, but for which its inverse is not terminating. In the world of finite ARSs such a rewriting system does not exist: if the system is terminating then it does not admit a cycle, and then also its inverse is terminating. But as soon as we allow a unary symbol in our rewriting system the situation is different: the single rewrite rule $f(a) \to a$ is terminating, while its inverse $a \to f(a)$ admits an infinite reduction.

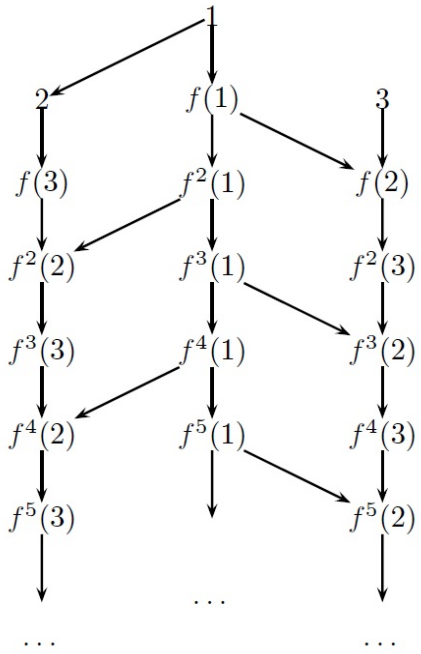
The goal of the current note is to automatically find examples in this richer class of rewriting systems. For the underlying machinery this is a much harder job: instead of dealing with only finitely many elements, now we have infinitely many terms, and then describing properties like termination and confluence in a SAT formula is much harder. To keep it feasible we decided to restrict to a quite limited class of term rewriting systems (TRSs): we only consider ground TRSs over constants and a single unary operation symbol. On the one hand this looks quite restrictive, on the other hand it is rich enough to cover the behavior just sketched: this class covers the single rewrite rule $f(a) \to a$ which is terminating but for which its inverse is not. We even make a further restriction: in the basic TRSs we are looking for, we only allow rules of the shape $a \to b$, $f(a) \to b$ and $a \to f(b)$, where $a, b$ are constants and $f$ is the unary symbol.

Our leading example is finding a TRS which is locally confluent but not confluent, and for which the inverse is terminating. In the world of finite ARSs this does not have a solution: if for such a finite ARS the inverse is terminating then the system itself is terminating, and then by Newman's Lemma local confluence implies confluence, see e.g. [1, 5]. But our prototype tool `Carpa+` based on the techniques of this note finds the following example satisfying the given properties fully automatically:

```
1 -> 2
1 -> f(1)
2 -> f(3)
3 -> f(2)
```

In fact, the above text is the literal output of `Carpa+`.

Indeed, this TRS is locally confluent as is easily checked by finding the common reduct $f(f(2))$ for the only critical pair $[2, f(1)]$. Its inverse is terminating since this inverse consists of the single rule $2 \to 1$ and three rules all removing a symbol $f$. The TRS has the following reduction graph:




From this reduction graph we easily see that it is not confluent: 1 rewrites both to 2 in the left column and to $f(2)$ in the right column, while 2 and $f(2)$ have no common reduct since all reducts of 2 are in the left column and all reducts of $f(2)$ are in the right column.

For main properties like confluence and local confluence we did not succeed in describing them exactly, in the sense of creating a formula that is satisfiable if and only if the desired property holds. For ground TRSs it is known that confluence is decidable, see e.g. [2]. However, for encoding confluence in satisfiability the corresponding algorithm is not suitable. Instead for the confluence related properties we deal with approximations. A main problem is to describe the relation $\to_R^*$: this can typically not be described as a single step with respect to a finite TRS. For instance, if $R$ contains the rule $f(a) \to a$, then $f^n(a) \to_R^* a$ for every $n$. In our approach we represent a subset $\to_R^*$ by finitely many rewrite rules, and use this to approximate $\to_R^*$ in the definition of local confluence. In this way we succeed in describing a property being slightly stronger than local confluence, and if local confluence is required instead we require this slightly stronger property, by which the approach remains sound.

For requiring non-confluence we need an approximation in the other direction. Here we project a TRS to a finite ARS by identifying $f^k(a)$ and $f^n(a)$ for all $k, n$ that are equal modulo

$m$ for some given number $m$, typically $m = 2$. We prove that confluence is preserved by projection. So instead of requiring non-confluence it is sound to require non-confluence of the finite ARS obtained by projection, and then for describing non-confluence of this finite ARS we exploit the techniques of [7].

For termination we succeed in giving an exact description: for a ground TRS over constants and a single unary symbol we succeed in finding a formula that is satisfiable if and only if the TRS is terminating. This formula is not a propositional formula in boolean variables, but involves real valued variables and linear inequalities among them. So instead of SAT we now need SMT (satisfiability modulo theories), for the theory consisting of linear inequalities. Fortunately, current tools like `yices` easily deal with this kind of SMT, so that is what we use in our approach.

This note is an extended abstract of the full paper [6]. For the proofs of theorems we refer to this full paper.

## 2 Theory

We fix a number $n$ and define $A = \{1, 2, \ldots, n\}$. We choose the signature $\Sigma = \{f\} \cup A$, where $f$ has arity 1 and all elements of $A$ have arity zero. Let $\mathcal{T}$ be the set of all ground terms over $\Sigma$, that is, the set of terms of the shape $f^i(a)$ for $i \geq 0$ and $a \in A$.

For $i, j \geq 0$ and $X \subseteq A \times A$ write $R_{ijX}$ for the ground TRS over $\Sigma$ consisting of the rules $f^i(a) \to f^j(b)$ for $(a, b) \in X$.

We define the ground TRS $T_1$ over $\Sigma$ to consist of all rules of the shape $a \to b$, $a \to f(b)$ and $f(a) \to b$ for $a, b \in A$, so $T_1 = R_{00A^2} \cup R_{01A^2} \cup R_{10A^2}$. Note that $T_1$ has $3n^2$ rules. Every subTRS of $T_1$ can be written as $R(X_{00}, X_{01}, X_{10}) = R_{00X_{00}} \cup R_{01X_{01}} \cup R_{10X_{10}}$ for three sets $X_{00}, X_{01}, X_{10} \subseteq A \times A$.

Similarly, we define the ground TRS $T_2$ to consist of the $6n^2$ rules of the shape $a \to b$, $a \to f(b)$, $f(a) \to b$, $f(a) \to f(b)$, $a \to f(f(b))$ and $f(f(a)) \to b$, for $a, b \in A$, so $T_2 = R_{00A^2} \cup R_{01A^2} \cup R_{10A^2} \cup R_{11A^2} \cup R_{02A^2} \cup R_{20A^2}$. SubTRSs of $T_2$ can be written as

$$R(X_{00}, X_{01}, X_{10}, X_{11}, X_{02}, X_{20}) \;=\; R_{00X_{00}} \cup R_{01X_{01}} \cup R_{10X_{10}} \cup R_{11X_{11}} \cup R_{02X_{02}} \cup R_{20X_{20}}$$

for six sets $X_{00}, X_{01}, X_{10}, X_{11}, X_{02}, X_{20} \subseteq A \times A$. Observe that $T_1 \subseteq T_2$ and $R(X_{00}, X_{01}, X_{10}) = R(X_{00}, X_{01}, X_{10}, \emptyset, \emptyset, \emptyset)$.

For TRSs $R = R(X_{00}, X_{01}, X_{10}, X_{11}, X_{02}, X_{20})$ and $S = R(X'_{00}, X'_{01}, X'_{10}, X'_{11}, X'_{02}, X'_{20})$, both subTRSs of $T_2$, we define $\mathsf{comp}(R, S) = R(Y_{00}, Y_{01}, Y_{10}, Y_{11}, Y_{02}, Y_{20})$ for $Y_{00}$, $Y_{01}$, $Y_{10}$, $Y_{11}$, $Y_{02}$, $Y_{20}$ defined by

$$\begin{array}{lcl}
Y_{00} &=& X_{00} \cdot X'_{00} \cup X_{01} \cdot X'_{10} \cup X_{02} \cdot X'_{20} \\
Y_{01} &=& X_{00} \cdot X'_{01} \cup X_{01} \cdot X'_{00} \cup X_{01} \cdot X'_{11} \cup X_{02} \cdot X'_{10} \\
Y_{10} &=& X_{00} \cdot X'_{10} \cup X_{10} \cdot X'_{00} \cup X_{11} \cdot X'_{10} \cup X_{01} \cdot X'_{20} \\
Y_{11} &=& X_{10} \cdot X'_{01} \cup X_{00} \cdot X'_{11} \cup X_{11} \cdot X'_{00} \cup X_{11} \cdot X'_{11} \cup X_{00} \cdot X'_{00} \cup X_{01} \cdot X'_{10} \cup X_{02} \cdot X'_{20} \\
Y_{02} &=& X_{01} \cdot X'_{01} \cup X_{00} \cdot X'_{02} \cup X_{02} \cdot X'_{00} \cup X_{02} \cdot X'_{11} \\
Y_{20} &=& X_{10} \cdot X'_{10} \cup X_{00} \cdot X'_{20} \cup X_{20} \cdot X'_{00} \cup X_{11} \cdot X'_{20}.
\end{array}$$

Define

$$\mathsf{inv}(R) = \{r \to \ell \mid \ell \to r \in R\} \quad \text{and} \quad \mathsf{rc}(R) = R \cup \{a \to a \mid a \in A\}.$$

**Theorem 1.** *Let $R \subseteq T_1$, $R_1 = \mathsf{rc}(R)$, $R_{i+1} = \mathsf{comp}(R_i, R_i)$ for $i > 0$, and*

$$\mathsf{comp}(\mathsf{inv}(R), R) \;\subseteq\; \mathsf{comp}(R_i, \mathsf{inv}(R_i))$$

*for some $i > 0$. Then $R$ is locally confluent.*

For Theorem 1 the requirement $R \subseteq T_1$ is essential: $R = \{a \to f(f(b)), a \to f(f(c))\}$ is not locally confluent but $\mathsf{comp}(\mathsf{inv}(R), R) \subseteq \mathsf{comp}(R_i, \mathsf{inv}(R_i))$ holds.

The local confluence criterion of Theorem 1 is sufficient, but not necessary. As an example consider $R = \{a \to f(b), f(a) \to c, b \to f(c), c \to f(c)\} \subseteq T_1$. This TRS is locally confluent as the only critical pair $[c, f(f(b))]$ converges via $c \to^3 f^3(c) \leftarrow f^2(b)$. However, $c \to f(f(b)) \in \mathsf{comp}(\mathsf{inv}(R), R)$, but $c \to f(f(b))$ is not in $\mathsf{comp}(R_i, \mathsf{inv}(R_i))$ for any $i$, since the convergence of the critical pair requires $c \to_{R_i} f^3(c)$, which is impossible for $R_i \subseteq T_2$.

We failed to give a full characterization of local confluence and only could give the sufficient criterion from Theorem 1. Next we show that by using functions from ground terms to real numbers, we can give a full characterization of termination of any finite ground TRS over $\Sigma = \{f\} \cup A$. This characterization is expressed as the existence of a number of real values satisfying a number of linear inequalities, so it is a feasibility requirement in linear programming. Where for local confluence the requirements from Theorem 1 were in propositional satisfiability (SAT), here we obtain a formula in a richer theory: SMT (satisfiability modulo theories), for the theory consisting of linear inequalities.

**Theorem 2.** *A ground TRS $R$ over $\{f\} \cup A$ for $A$ finite is terminating if and only if a map $W : A \to \mathbf{R}$ exists such that $W(a) + n > W(b) + k$ for every $f^n(a) \to f^k(b) \in R$.*

Choosing reals rather than integers in Theorem 2 is essential: $\{a \to b,\ f(b) \to a\}$ is terminating but does not allow an integer valued weight function $W$ satisfying $W(a) > W(b)$ and $W(b) + 1 > W(a)$.

Next we investigate how a TRS can be projected to a finite ARS in such a way that some properties preserve. In particular, the projection of a confluent TRS is again confluent, so if the projection is not confluent, we can conclude that neither the original TRS is confluent.

Fix an integer $m > 1$. For a signature $\Sigma = \{f\} \cup A$, where $f$ has arity 1 and all elements of the finite set $A$ have arity zero, we define a new signature $\Sigma' = \{a_i \mid a \in A \wedge 0 \leq i < m\}$, in which all elements of $\Sigma'$ are constants. For a ground TRS $R$ over $\Sigma$ we define the finite ARS $R'$ on $\Sigma'$ by defining $a_i \to_{R'} b_j$ if and only if there exists a rule $f^n(a) \to f^k(b)$ in $R$ for which $i - j \equiv n - k \mod m$.

**Theorem 3.** *In the above setting, if $R$ is confluent, then $R'$ is confluent too.*

# 3 Implementation

In earlier work [7] we developed a tool `Carpa` reading a list of desired properties in a corresponding input language, looking for finite ARSs. In this input language one can specify the number of ARSs one is looking for, the number of elements on which these ARSs act, several constructions to combine ARSs like union, composition, peaks and valleys, and properties like (local) confluence and termination. When `Carpa` is executed on such an input then it tries to construct a corresponding set of finite ARSs satisfying all specified properties, and in case of success it yields these ARSs as output.

Now we developed a prototype tool `Carpa+` that does the same for a simple class of TRSs rather than finite ARSs, namely the class of subTRSs of $T_1$, exploiting the theory as presented in this paper. As some constructions are specific for TRSs, like the projection to finite ARSs, we decided to keep the input language for `Carpa+` closely related to that for `Carpa`, but not exactly the same. For instance, `Carpa+` admits an operation `mod2` to project a TRS to a finite

ARS as described above for $m = 2$, which is not in `Carpa`. Conversely, `Carpa` admits operations like `tc` for transitive closure which are not in `Carpa+`. Just like `Carpa`, also `Carpa+` can be downloaded in zip format from

`http://www.win.tue.nl/~hzantema/carpa.html`

including the source code, a Linux executable and several examples. For instance, after extracting the zip file in a directory in a Linux environment, by calling `./carpa+ ex10`, the tool is applied on the leading example of this note, and the output as announced in the introduction is generated within a fraction of a second. Internally, here for local confluence, termination and non-confluence the implementation follows the encodings from Theorems 1, 2 and 3, respectively.

# 4 Conclusion

In earlier work [7] we described how to find finite ARSs fully automatically satisfying a given list of properties, based upon transforming the requirements to a SAT problem. This work was inspired and initiated by [3, 4]. In this note we did some first steps to extend this work to automatically finding TRSs rather than finite ARSs. This turned out to be a hard job: many questions become much harder when lifting a setting of finitely many elements to a setting of infinitely many terms. Restricting to a very limited class of term rewriting systems we developed a prototype tool `Carpa+` in which the requirements are entered, a corresponding formula is composed, the SMT solver `yices` is called on this formula, and the output is inspected to obtain the desired result for our leading example. This approach also applies on some variants and other properties, but we fully realize that as a first step this is more a case study than a general tool for finding TRSs with a given list of properties. Next steps in this direction would include both extension to richer classes of TRSs and better approximations of the basic rewriting properties.

# References

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

[2] B. Felgenhauer. Deciding Confluence of Ground Term Rewrite Systems in Cubic Time. In Ashish Tiwari, editor, *23rd International Conference on Rewriting Techniques and Applications (RTA'12)*, volume 15 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 165–175, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[3] A. Stump, G. Kimmell, and R. El Haj Omar. Type Preservation as a Confluence Problem. In Manfred Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA)*, volume 10 of *LIPIcs*, pages 345–360, 2011.

[4] A. Stump, G. Kimmell, H. Zantema, and R. El Haj Omar. A rewriting view of simple typing. *Logical Methods in Computer Science*, 9(1), 2012.

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

[6] H. Zantema. Automatically finding particular term rewriting systems. Available via `http://www.win.tue.nl/~hzantema/carpa.html`, 2013.

[7] H. Zantema. Finding small counter examples for abstract rewriting properties. *Mathematical Structures in Computer Science*, 2013. Accepted, preliminary version available via `http://www.win.tue.nl/~hzantema/carpa.html`.

# Confluent Unfolding in the $\lambda$-calculus with letrec

Jan Rochel[1] and Clemens Grabmayer[2]

[1] Department of Information and Computing Sciences, Utrecht University, The Netherlands
[2] Department of Philosophy, Utrecht University, The Netherlands

**Abstract**

We show that a rewriting system for unfolding terms in the $\lambda$-calculus with letrec is confluent. This system is from previous work, where we formulate letrec-unfolding as a Combinatory Reduction System (CRS). We prove confluence by applying the decreasing diagrams method to a partitioning of the parallel rewriting relation into relations that are induced by parallel steps in which a given rule contracts redexes at a given letrec-depth.

In [1] (see also [2])[1] we study infinite $\lambda$-terms and present two characterisations of those $\lambda$-terms that are expressible in the $\lambda$-calculus with letrec ($\lambda_{\mathsf{letrec}}$), in the sense that they can be obtained as the infinite unfoldings of a $\lambda_{\mathsf{letrec}}$-term. One characterisation is by a structural analysis of the term: a term is $\lambda_{\mathsf{letrec}}$-expressible if and only if there it has no infinite 'binding–capturing chains'. The other characterisation is via the concept of 'strong regularity': a term is $\lambda_{\mathsf{letrec}}$-expressible if and only if it is strongly regular.

We define a Combinatory Reduction System (CRS) for unfolding $\lambda_{\mathsf{letrec}}$-terms. In the paper confluence of the CRS is important since it guarantees that the unfolding of a term is unique. We think however, that the proof itself is of independent interest. For simplicity we use in this extended abstract an informal formulation of $\lambda_{\mathsf{letrec}}$-terms and the unfolding rewriting system instead. The set of $\lambda_{\mathsf{letrec}}$-terms $Ter(\boldsymbol{\lambda}_{\mathsf{letrec}})$ is inductively defined by the following grammar:

$$\begin{array}{llcll} (\textit{term}) & L & ::= & \lambda x.L & (\textit{abstraction}) \\ & & | & L\,L & (\textit{application}) \\ & & | & x & (\textit{variable}) \\ & & | & \mathsf{letrec}\ B\ \mathsf{in}\ L & (\textit{letrec}) \\ (\textit{binding group}) & B & ::= & f_1 = L\ \ldots\ f_n = L & (\textit{equations}) \end{array}$$

On this set we describe letrec-unfolding in the rewriting system $\boldsymbol{R}_{\triangledown}$ as follows. The names of the first four rules are chosen to reflect the kind of term that resides directly inside of the in-part of the letrec-term, which helps to see that the rules are complete in the sense that every term of the form $\mathsf{letrec}\ B\ \mathsf{in}\ L$ is a redex.

$$\begin{array}{rrcl} (\varrho^{@}_{\triangledown}): & \mathsf{letrec}\ B\ \mathsf{in}\ L_0\,L_1 & \to & (\mathsf{letrec}\ B\ \mathsf{in}\ L_0)\,(\mathsf{letrec}\ B\ \mathsf{in}\ L_1) \\ (\varrho^{\lambda}_{\triangledown}): & \mathsf{letrec}\ B\ \mathsf{in}\ \lambda x.L_0 & \to & \lambda x.\mathsf{letrec}\ B\ \mathsf{in}\ L_0 \\ (\varrho^{\mathrm{letrec}}_{\triangledown}): & \mathsf{letrec}\ B_0\ \mathsf{in}\ \mathsf{letrec}\ B_1\ \mathsf{in}\ L & \to & \mathsf{letrec}\ B_0, B_1\ \mathsf{in}\ L \\ (\varrho^{\mathrm{rec}}_{\triangledown}): & \mathsf{letrec}\ B\ \mathsf{in}\ f_i & \to & \mathsf{letrec}\ B\ \mathsf{in}\ L_i \quad (\text{if } B \text{ is } f_1 = L_1 \ldots f_n = L_n) \\ (\varrho^{\mathrm{nil}}_{\triangledown}): & \mathsf{letrec}\ \mathsf{in}\ L & \to & L \\ (\varrho^{\mathrm{red}}_{\triangledown}): & \mathsf{letrec}\ f_1 = L_1 \ldots f_n = L_n\ \mathsf{in}\ L & \to & \mathsf{letrec}\ f_{j_1} = L_{j_1} \ldots f_{j_{n'}} = L_{j_{n'}}\ \mathsf{in}\ L \\ & & & (\text{if } f_{j_1}, \ldots, f_{j_{n'}} \text{ are the recursion variables reachable from } L) \end{array}$$

'Reachable' from $L$ in the last rule refers to recursion variables that either occur in $L$ or on the right hand side of any equation that is reachable from $L$. Thus, the condition on the rule ensures that only superfluous equations are removed from the binding group.

[1] In [2] (to appear) we consider the simpler case of expressibility in the $\lambda$-calculus with $\mu$.

**Theorem.** $\boldsymbol{R}_{\triangledown}$ is confluent.

*Proof sketch.* First all, we cannot use Newman's Lemma to prove the theorem, since $\boldsymbol{R}_{\triangledown}$ is not terminating. To show confluence of $\boldsymbol{R}_{\triangledown}$ we use the method of 'decreasing diagrams' [4, Sec. 2.3] [3, Sec. 14.2]. We use it however not to prove confluence of $\boldsymbol{R}_{\triangledown}$ directly, but of the abstract reduction system $\mathcal{A} = (Ter(\boldsymbol{\lambda}_{\mathsf{letrec}}), \{\mathrel{-\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow}_{\rho_d} \mid (d,\rho) \in \mathbb{N} \times R\})$ with $R$ as the set of rules of $\boldsymbol{R}_{\triangledown}$ where $\mathrel{-\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow}_{\rho_d}$ denotes the parallel rewriting relation on $Ter(\boldsymbol{\lambda}_{\mathsf{letrec}})$ induced by rule $\rho$ at letrec-depth $d$. As a precedence order we take the order induced by the letrec-depth:

$$\rho_d \geq \sigma_{d'} \iff d \geq d'$$

The letrec-depth of a redex in $\lambda_{\mathsf{letrec}}$-term denotes the number of letrec-nodes passed on the path from the root of the term tree to the corresponding position. We write $\rightarrow_{\rho_d}$ to denote the relation induced by applying rule $\rho$ contracting a redex at letrec-depth $d$.

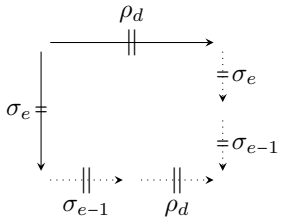
Let us denote the rewriting relation induced by $\mathcal{A}$ by $\rightarrow_{\mathcal{A}}$:

$$\rightarrow_{\mathcal{A}} = \bigcup \{\mathrel{-\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow}_{\rho_d} \mid (d,\rho) \in \mathbb{N} \times R\}$$

If $\rightarrow_{\mathcal{A}}$ is confluent then the rewriting relation $\rightarrow_{\triangledown}$ induced by $\boldsymbol{R}_{\triangledown}$ is confluent because it holds: $\rightarrow_{\triangledown} \subseteq \rightarrow_{\mathcal{A}} \subseteq \twoheadrightarrow_{\triangledown}$ or equivalently $\twoheadrightarrow_{\mathcal{A}} = \twoheadrightarrow_{\triangledown}$ (see also [4, Lemma 2.2.5]).

We use the approach with the parallel steps because the preceding attempt to prove confluence of $\boldsymbol{R}_{\triangledown}$-steps by decreasing diagrams more directly was unsuccessful. As a precedent order we considered an ordering on the rules and lexicographic extensions of such orderings with the letrec-depth of the contracted redex. We came to the conclusion that no such order could ensure decreasingness of the elementary diagrams of both the critical pairs as well as the strictly nested redexes. This was due to redex duplication induced by the diverging steps, so that joining the diagram required a multi-step that disrupted decreasingness. In order to resolve this problem we considered parallel steps as above such that the problematic multi-step would become a single parallel step. This led to more intricate diagrams but turned out to be a viable solution.

We will prove confluence of $\rightarrow_{\mathcal{A}}$ by showing that two diverging parallel steps in $\boldsymbol{R}_{\triangledown}$ can be joined in an elementary diagram of the following form with $d \leq e$.

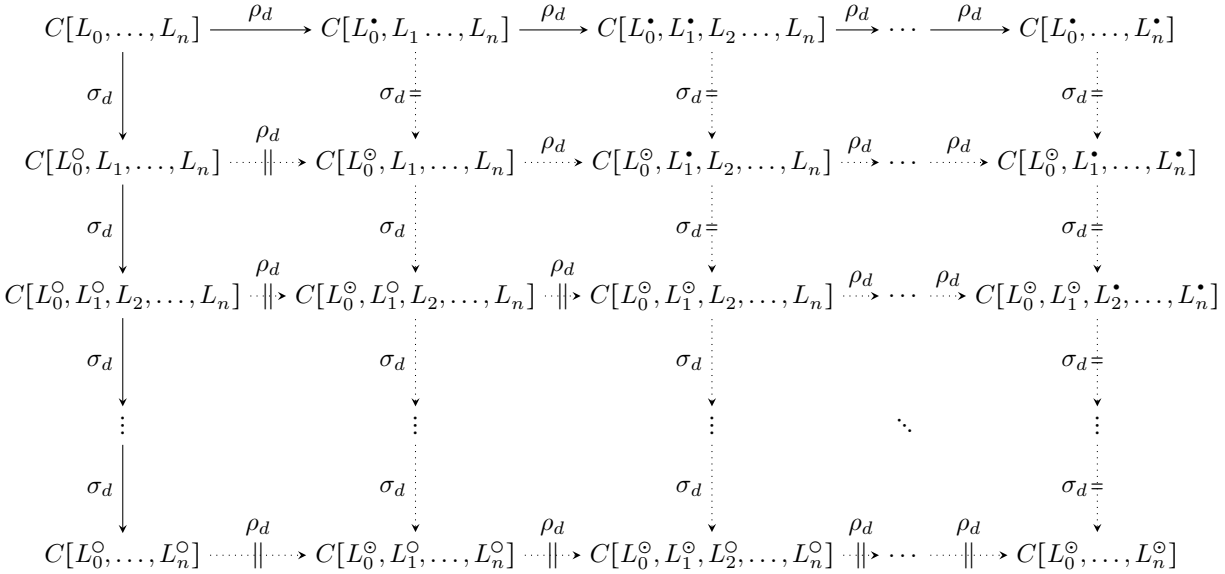



(1)

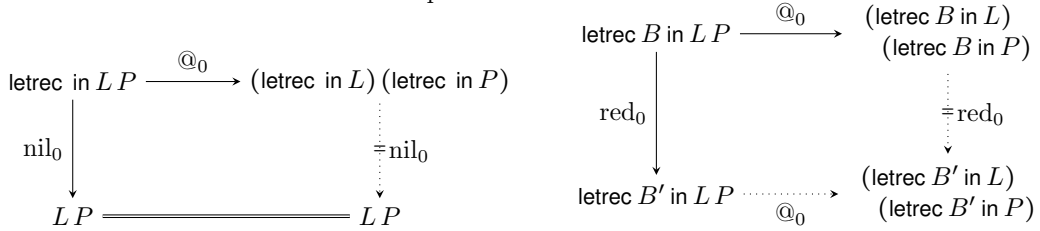
With the precedence as above the diagram is decreasing. Note that in all the diagrams we implicitly assume the reflexive closure for all arrows. The rest of the proof is structured as follows. To justify the diagram we distinguish the cases $d = e$ and $d < e$, for which we construct diagrams that are instances of (1).

**Case 1.** For $d = e$ we need to consider parallel diverging steps contracting redexes at the same letrec-depth $d$. We construct the diagram below which is an instance of the diagram above where the diverging parallel steps are in sequentialised form. We write terms as fillings of a multihole context $C$ with all its holes at letrec-depth $d$ such that the contracted $\rho_d$- and $\sigma_d$-redexes are filled into these holes. In this way we can make explicit at which position a step takes place, i.e.

at the root of the context hole fillings. The topmost row and the leftmost column are respective sequentialisations of the parallel diverging $\rho_d$- and $\sigma_d$-steps into single steps.

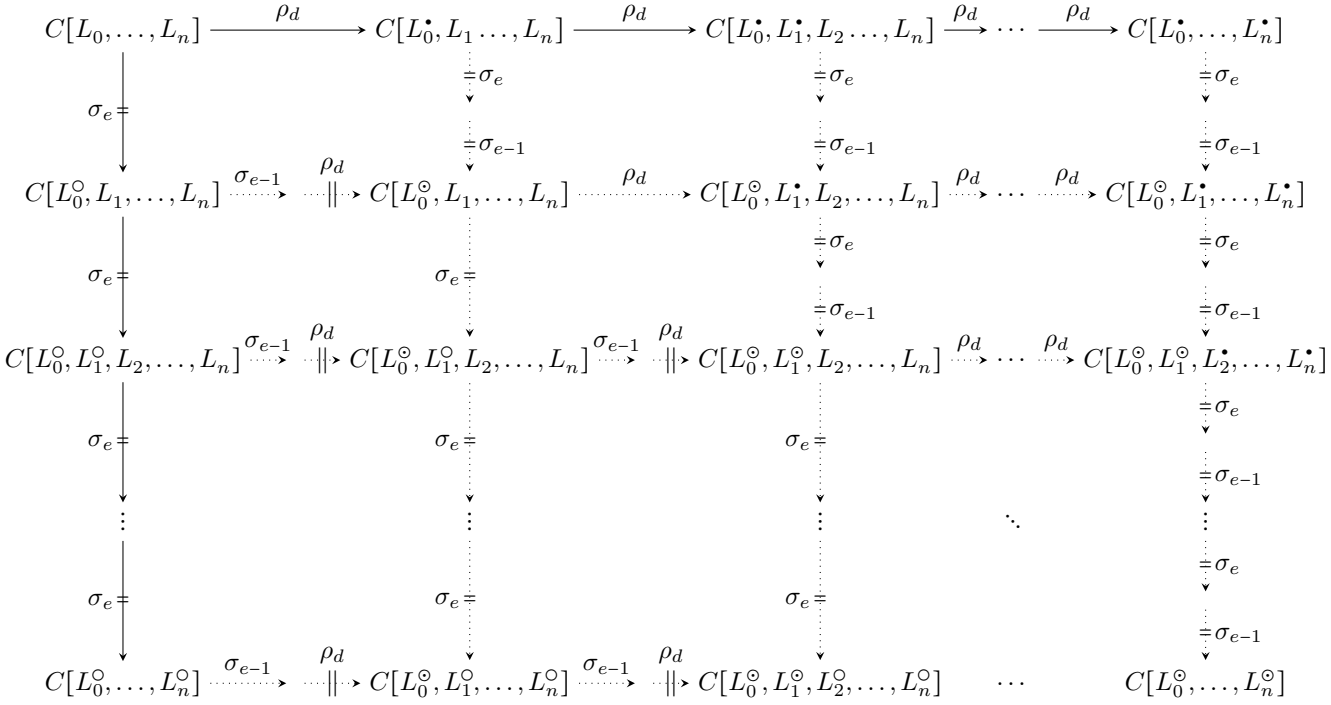


Only the tiles on the diagonal require closer attention because for all other tiles the vertical and horizontal steps take place in different holes of the context, therefore they are disjoint and consequently commute. In the tiles on the diagonal the diverging steps may be either due to a critical pair or to identical steps. In the latter case the diagram is easily joined. In case of a critical pair, since all steps take place at the same letrec-depth any such critical pair must arise from a root overlap. An exhaustive scrutiny of all these critical pairs reveals that they can be joined in a way that conforms to the tiles on the diagonal. Below two exemplary cases are shown. Note that the letrec-depths of the steps have to be increased by $d$ according to the lifting into a context with its hole at letrec-depth $d$.

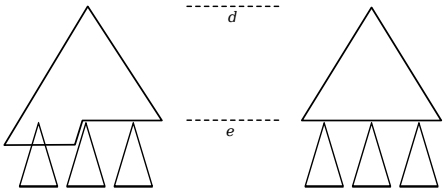


**Case 2.** For $d < e$ we use the same approach as for $d = e$, the diagram is however more involved. Again, we use a context $C$ with context holes at letrec-depth $d$. But since $e > d$, more than one $\sigma_e$-contraction may take place in one such hole. Therefore a per-hole partitioning of the vertical steps requires a sequence of parallel steps.

The diagram below fits the scheme of the elementary diagram (1) when interleaving the $\sigma_e$-steps with the $\sigma_{e-1}$-steps in the rightmost column such that steps at depth $e$ preceed those at depth $e-1$. Similarly for the bottommost row where the $\rho_{e-1}$-steps have to preceed the $\sigma_d$-steps. These reorderings are possible since the segments represent contractions within different holes of $C$. As in the previous diagram the tiles which do not lie on the diagonal are unproblematic, which leaves us to complete the proof by constructing the tiles on the diagonal.

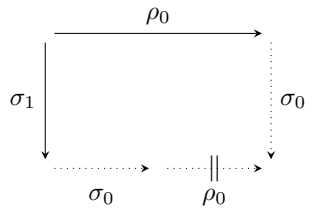


Every hole on the diagonal is filled with at most one $\rho_d$-redex (at the root of the context hole fillings) but because of $d < e$ with possibly many $\sigma_e$-redexes (properly inside of the fillings). There may or may not be an overlap between the $\rho_d$-step and a $\sigma_e$-step, but there can be at most one, which is due to the rules of $\boldsymbol{R}_{\triangledown}$.

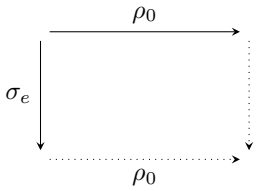



Therefore $\sigma_e$ contracts either an overlap and a number of nested redexes, or only nested redexes without an overlap. These constellations are depicted on the figure on the left. There is one $\rho_d$-redex and three $\sigma_e$-redexes. On the left, one of the $\sigma_e$-redexes overlaps with the $\rho_d$-redex while on the right all $\sigma_e$-redexes are strictly nested inside the $\rho_d$-redex.

For the critical pairs due to a non-root overlap, and for all situations with nested redexes, we construct diagrams of the following shape, respectively:

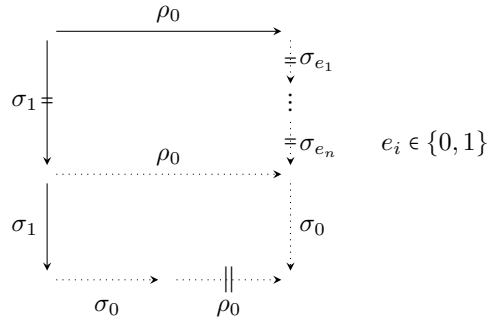



(2)




$e' \in \{e, e-1\}$ (3)

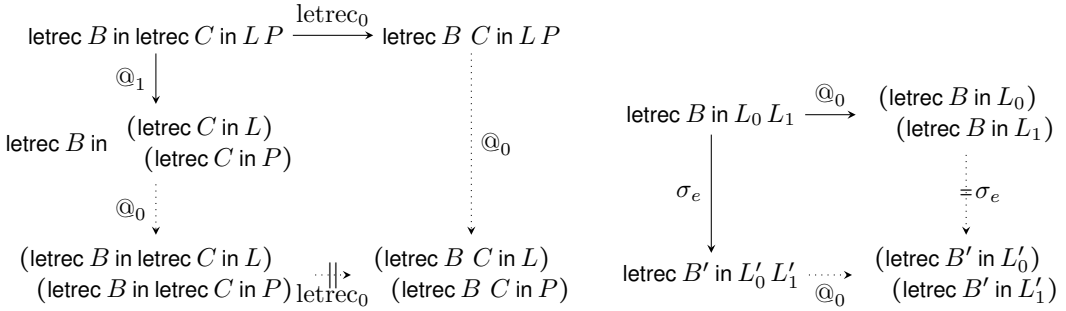
When lifted into a context of letrec-depth $d$ both of the diagrams comply to the shape necessary for the diagonal tiles, but we need to be able to handle situations as on the left of the above figure, where both nested redexes as well as the overlapping redex are contracted. Firstly, since all $\sigma$-redexes occur at the same letrec-depth, it must hold that $d = 0$ and $e = 1$, which is due to the rules of $\boldsymbol{R}_{\triangledown}$. Secondly, none of the involved redex contractions affect any of the nested redexes except for duplicating or erasing them, which means that the residuals of the $\sigma$-steps after these steps are part of a parallel $\sigma_{e'}$-step (mind that we assume the reflexive closure of all steps). Or in a diagram:




The diagram is composed from the previous two diagrams. A parallel version of (3) constitutes the top part, while the bottom part is an exact replica of (2). The top part settles the portion arising from the nested redexes, the bottom part settles the portion arising from the overlapping redex.

At last in order to fit that diagram into the scheme of the diagonal tiles the steps on the right have to be reordered such that $\sigma_{e_i}$-steps with $e_i = 1$ preceed $\sigma_{e_i}$-steps with $e_i = 0$. The reordering is viable because every $\sigma_{e_i}$-step takes place in its own residual of the $\sigma_1$-step from the left.

We conclude the proof by a comprehensive analysis of all critical pairs that arise from non-root overlaps in $\boldsymbol{R}_{\triangledown}$ as well as the diagrams for joining nested redexes. Below, one critical pair is shown for each case. See [1] for an exhaustive scrutinisation.




□

A generalisation of the proof to obtain a theorem is still very much work in progress. Below are preliminary propositions that are to capture the essential properties of $\boldsymbol{R}_{\nabla}$ that made the above approach possible.

The following lemma requires the rewriting steps to be partitioned in a way such that diverging parallel steps cannot be 'intertwined', permitting the construction of a decreasing diagram using parallel steps as in the proof above.

**Lemma 1.** *Let $\mathcal{A}$ be an ARS $(A, \{\to_\alpha \mid \alpha \in I\})$ that is induced by a TRS/CRS/HRS, where the index set $I$ is equipped with a well-founded partial order. The steps $\Phi$ of $\mathcal{A}$ are equipped with respective indices from $I$ as well as with the position ($pos : \Phi \to \mathbb{N}$) of the contracted redex. Indexed single steps $\to_\alpha$ induce indexed parallel steps $\Rightarrow_\alpha$ (for all $\alpha \in I$). If the following conditions are met, then $\to_I = \bigcup_{i\in I} \to_i$ is confluent.*

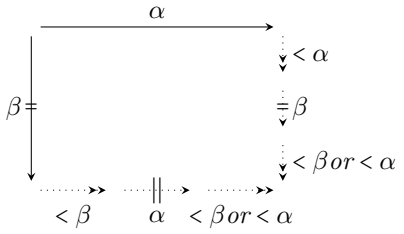
1. *There do not exist $\alpha, \beta \in I$, two $\alpha$-steps $\rho_1$, $\rho_2$, and two $\beta$-steps $\sigma_1$, $\sigma_2$ such that it holds:*

$$pos(\rho_1) \parallel pos(\rho_2) \wedge pos(\sigma_1) \parallel pos(\sigma_2)$$

$$pos(\rho_1) < pos(\sigma_1) \wedge pos(\rho_2) > pos(\sigma_2)$$

   *Here we use $\parallel$ for incomparable ('parallel') positions: $p \parallel q \Leftrightarrow p \not\leq q \wedge q \not\leq p$.*

2. *A diverging $\alpha$-step at position $p$ and a parallel $\beta$-step with positions $q_1, \ldots, q_n$ below $p$ ($\forall i \in \{1, \ldots, n\} : p \leq q_i$) can be joined by a diagram of the following form:*




   *Thereby all closing steps need to take place below $p$, i.e. at positions $\geq p$.*

A second specialised lemma is to be more concrete and easier to apply. It includes in the index a notion of depth (cf. letrec-depth), to which the order on the index is linked, such that condition 1 of Lemma 1 is met. Furthermore we stipulate properties of the rewriting relation, that allow for an order-respecting context embedding of rewriting steps. This will simplify condition 2 of Lemma 1 such that the diagram has only to be constructed for critical overlaps at the root of a term.

# References

[1] Clemens Grabmayer and Jan Rochel. Expressibility in the Lambda Calculus with Letrec. Technical report, August 2012. `http://arxiv.org/abs/1208.2383`.

[2] Clemens Grabmayer and Jan Rochel. Expressibility in the Lambda Calculus with Mu. In *Proceedings of RTA 2013*, 2013. To appear.

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

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

# Rule Labeling for Confluence of Left-Linear Term Rewrite Systems*

Bertram Felgenhauer

Institute of Computer Science, University of Innsbruck, Austria
bertram.felgenhauer@uibk.ac.at

**Abstract**

Rule labeling is a heuristic, suggested by van Oostrom, for applying decreasing diagrams to linear TRSs. The heuristic also works for left-linear TRSs under certain relative termination conditions. In this note we apply the rule labeling heuristic to arbitrary left-linear TRSs, based on considering parallel reduction relations and parallel critical peaks.

## 1 Introduction

The decreasing diagrams technique by van Oostrom [6] is a powerful criterion for showing confluence of abstract rewrite systems. In [7], van Oostrom suggested the rule labeling heuristic for establishing confluence of linear term rewrite systems. The heuristic consists of labeling each rewrite step with the used rule. Confluence can be established by showing that all critical pairs can be joined decreasingly. Rule labeling can also be applied to certain duplicating left-linear TRSs, by combining it lexicographically with other labelings [7, 3, 9].

In this note we revisit the rule labeling heuristic and show how it can be applied to arbitrary left-linear TRSs, if parallel reduction and parallel critical pairs are considered.

The paper is organized as follows. Section 2 is devoted to preliminaries. Then, in Section 3, we present a confluence criterion for left-linear systems based on rule labeling. In Section 4 we sketch how this idea extends to weak ll-labelings from [9]. Finally, we conclude in Section 5.

## 2 Preliminaries

We assume that the reader is familiar with standard term rewriting terminology [1, 5].

Given a rewrite relation $\to$, we use $\xrightarrow{=}$ and $\xrightarrow{*}$ to denote its reflexive closure and reflexive, transitive closure, respectively. The space below the arrows is reserved for labels: $\underset{\alpha}{\to}$ denotes a rewrite step labeled with $\alpha$. Given a set of labels $\mathcal{L}$ with well-founded precedence $\succ$, and a family $(\underset{\alpha}{\to})_{\alpha\in\mathcal{L}}$, we let $\underset{\curlyvee\alpha}{\longrightarrow} = \bigcup_{\alpha\succ\beta}(\underset{\beta}{\to})$ and $\underset{\curlyvee\alpha\beta}{\longrightarrow} = \underset{\curlyvee\alpha}{\longrightarrow} \cup \underset{\curlyvee\beta}{\longrightarrow}$. A local peak $s \underset{\alpha}{\leftarrow} \cdot \underset{\beta}{\to} t$ is said to be joined decreasingly if $s \overset{*}{\underset{\curlyvee\alpha}{\longleftrightarrow}} \cdot \overset{=}{\underset{\beta}{\to}} \cdot \overset{*}{\underset{\curlyvee\alpha\beta}{\longleftrightarrow}} \cdot \overset{=}{\underset{\alpha}{\leftarrow}} \cdot \overset{*}{\underset{\curlyvee\beta}{\longleftrightarrow}} t$. If all local peaks can be joined decreasingly, then $\to = \bigcup_{\alpha\in\mathcal{L}}(\underset{\alpha}{\to})$ is confluent [7].

We denote parallel rewrite steps by $\twoheadrightarrow$. If we wish to indicate the set of positions involved in a parallel rewrite step, we write $\overset{P}{\twoheadrightarrow}$. For a set of positions $P$ and term $t$ we let $t|_P = \{t|_p \mid p \in P\}$. Parallel critical pairs [2] arise similarly to critical pairs: whereas we obtain a critical pair $l[r']_p\sigma \leftarrow\!\rtimes\!\rightarrow r\sigma$ whenever $p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(l)$ and $\sigma$ is a most general unifier of $l|_p$ and $l'$, where $l \to r$ and $l' \to r'$ are variants of rules in $\mathcal{R}$ with no common variables, a parallel critical pair is conceived as $l[r^p]_{p\in P}\sigma \twoheadleftarrow\!\rtimes\!\rightarrow r\sigma$ if $P \subseteq \mathcal{P}\mathsf{os}_{\mathcal{F}}(l)$ is a set of mutually parallel positions, $l \to r$,

*This research was supported by the Austrian Science Fund (FWF) project P22467-N23.

$l^p \to r^p$ are variants of rules in $\mathcal{R}$ with no common variables, and $\sigma$ is a most general solution to the set of equations $l|_p\sigma = l^p\sigma$ for $p \in P$. If $\mathcal{R}$ is finite, then the set of its parallel critical pairs is finite as well.

**Proposition 1.** *Let $\mathcal{R}$ be a left-linear TRS and $t \overset{P}{\twoheadleftarrow} s \overset{\epsilon}{\to} u$. Then there are substitutions $\sigma \twoheadrightarrow \sigma'$ and a parallel critical pair $t' \twoheadleftarrow\!\rightarrow u'$ such that $t = t'\sigma' \overset{P\setminus P'}{\twoheadleftarrow} t'\sigma \overset{P'}{\twoheadleftarrow} s \to u'\sigma = u$, where $P' \subseteq P$. (Note that left-linearity is essential for the substitutions $\sigma$ and $\sigma'$ to exist.)*

# 3 Confluence by Rule Labeling

Throughout this section we assume a given left-linear TRS $\mathcal{R}$, and a set of labels $\mathcal{L}$ equipped with a well-founded order $\succ$. Furthermore, let $\hat{\ell} : \mathcal{R} \to \mathcal{L}$ map rewrite rules to labels. The rule labeling heuristic labels rewrite steps according to the used rule, $\ell(t \to_{l\to r} t') = \hat{\ell}(l \to r)$.

**Definition 2.** Consider a parallel step $t \overset{P}{\twoheadrightarrow} t'$. For each $p \in P$, we have a rewrite step $t \to t[t'|_p]_p$ at position $p$. The set $\Gamma \subseteq \mathcal{L}$ is a valid label for $t \twoheadrightarrow t'$ if $\ell^{\parallel}(t \twoheadrightarrow t') \subseteq \Gamma$, where

$$\ell^{\parallel}(t \overset{P}{\twoheadrightarrow} t') = \{\ell(t \to t[t'|_p]_p) \mid p \in P\}$$

This means that a parallel rewrite step is labeled—at least—by the set of the labels of the rules used in the step. We indicate labels along with the step, writing $t \underset{\Gamma}{\twoheadrightarrow} t'$. A step $t \underset{\Gamma}{\twoheadrightarrow} t'$ is *homogeneous* if all its labels are the same, i.e., $\#\Gamma \leqslant 1$, and *heterogeneous* otherwise.

Parallel labels are ordered by the (multi-)set extension of $\succ$, which we also denote by $\succ$. The interest in homogeneous and heterogeneous steps stems from the following proposition:

**Proposition 3.** *If $s \underset{\Gamma}{\overset{P}{\twoheadrightarrow}} t$ is a heterogeneous step, i.e., $\#\Gamma > 1$, then we can split it into a sequence of individual steps whose labels are smaller than $\Gamma$ w.r.t. $\succ$.*

We are now ready to state and prove the main theorem of this section.

**Theorem 4.** *A left-linear TRS $\mathcal{R}$ is confluent if all its parallel critical peaks $t \underset{\Gamma}{\overset{P}{\twoheadleftarrow}} s \underset{\Delta}{\longrightarrow} u$ with homogeneous parallel step, i.e., $\#\Gamma = \#\Delta = 1$ can be joined decreasingly as*

$$t \underset{\succ\Gamma}{\overset{*}{\longrightarrow}} \cdot \underset{\Delta}{\twoheadrightarrow} \cdot \underset{\succ\Gamma\Delta}{\overset{*}{\longrightarrow}} \cdot \underset{\succ\Gamma\Delta}{\overset{*}{\longleftarrow}} v \underset{\Gamma}{\overset{Q}{\twoheadleftarrow}} \cdot \underset{\succ\Delta}{\overset{*}{\longleftarrow}} u$$

*such that $\mathcal{V}\mathsf{ar}(v|_Q) \subseteq \mathcal{V}\mathsf{ar}(s|_P)$.*

*Proof.* We show that $\twoheadrightarrow$ is locally decreasing, which implies confluence of $\mathcal{R}$. Let $t \underset{\Gamma}{\overset{P}{\twoheadleftarrow}} s \underset{\Delta}{\overset{Q}{\twoheadrightarrow}} u$, where $\Gamma, \Delta \subseteq \mathcal{L}$. If $\#\Gamma > 1$ then we can join $t$ and $u$ by replacing $s \twoheadrightarrow t$ by a sequence of smaller steps using Proposition 3. We obtain a conversion $s \underset{\succ\Gamma}{\overset{*}{\longleftarrow}} \cdot \underset{\Delta}{\twoheadrightarrow} u$, resulting in a locally decreasing diagram. The case $\#\Delta > 1$ is handled symmetrically. If $\Gamma = \varnothing$ or $\Delta = \varnothing$, then $t$ and $u$ can be joined by the other rewrite step, also resulting in a decreasing diagram.

In the remaining case, $\#\Gamma = 1$ and $\#\Delta = 1$, that is, we have a peak between homogeneous parallel steps steps. Let $\alpha \in \Gamma$ and $\beta \in \Delta$. Then $t \underset{\Gamma}{\overset{P}{\twoheadleftarrow}} s \underset{\Delta}{\overset{Q}{\twoheadrightarrow}} u$. It suffices to show that

$$t \underset{\succ\Gamma}{\overset{*}{\longrightarrow}} \cdot \underset{\Delta}{\twoheadrightarrow} \cdot \underset{\succ\Gamma\Delta}{\overset{*}{\longrightarrow}} \cdot \underset{\succ\Gamma\Delta}{\overset{*}{\longleftarrow}} \cdot \underset{\Gamma}{\twoheadleftarrow} \cdot \underset{\succ\Delta}{\overset{*}{\longleftarrow}} u \tag{1}$$

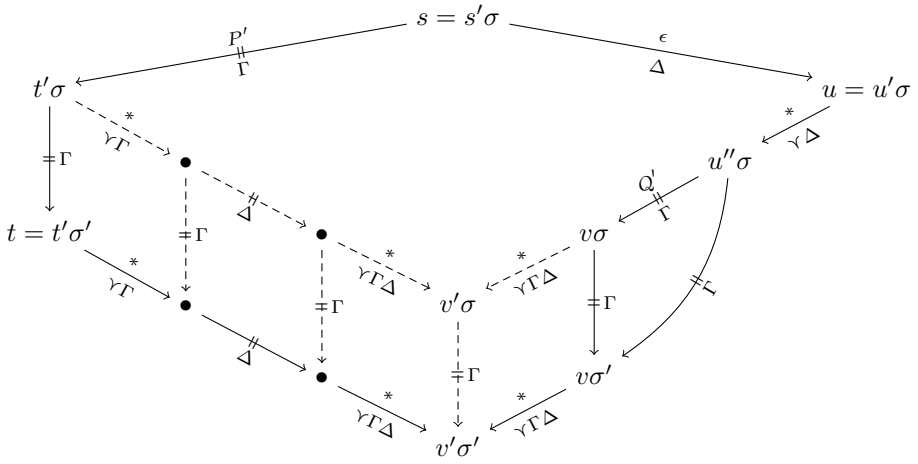



Figure 1: Proof of Theorem 4

We claim that (1) holds whenever $P = \{\epsilon\}$ or $Q = \{\epsilon\}$. Then for all $p \in \min(P \cup Q)$, $t \overset{P}{\underset{\Gamma}{\leftarrow\!\!\!\!\!\!\shortparallel}} s \overset{Q}{\underset{\Delta}{\shortparallel\!\!\!\!\!\!\rightarrow}} u$ induces a peak $t|_p \overset{P'}{\underset{\Gamma}{\leftarrow\!\!\!\!\!\!\shortparallel}} s|_p \overset{Q'}{\underset{\Delta}{\shortparallel\!\!\!\!\!\!\rightarrow}} u|_p$, where $P' = \{\epsilon\}$ or $Q' = \{\epsilon\}$. So for each $p$, we obtain a joining sequence for $t|_p$ and $u|_p$ of the shape (1). Since the positions in $\min(P \cup Q)$ are mutually parallel, these sequences for each $p \in \min(P \cup Q)$ can be combined into a single sequence of the same shape without any difficulties. In particular, the $\underset{\Delta}{\shortparallel\!\!\!\!\!\!\rightarrow}$ steps combine into a single $\underset{\Delta}{\shortparallel\!\!\!\!\!\!\rightarrow}$ step and likewise, the $\underset{\Gamma}{\leftarrow\!\!\!\!\!\!\shortparallel}$ steps can be combined into a single $\underset{\Gamma}{\leftarrow\!\!\!\!\!\!\shortparallel}$ step.

In order to show (1) for $P = \{\epsilon\}$ or $Q = \{\epsilon\}$, assume w.l.o.g. that $Q = \{\epsilon\}$. By Proposition 1, there are a parallel critical peak $t' \overset{P'}{\underset{}{\leftarrow\!\!\!\!\!\!\shortparallel}} s' \to u'$ and substitutions $\sigma$, $\sigma'$ such that $\sigma \underset{\Gamma}{\shortparallel\!\!\!\!\!\!\rightarrow} \sigma'$ and $t = t'\sigma' \overset{P \setminus P'}{\underset{\Gamma}{\leftarrow\!\!\!\!\!\!\shortparallel}} t'\sigma \overset{P'}{\underset{\Gamma}{\leftarrow\!\!\!\!\!\!\shortparallel}} s'\sigma = s \underset{\Delta}{\overset{\epsilon}{\rightarrow}} u'\sigma = u$, where $P' \subseteq P$. By assumption we can join $t'$ and $u'$ decreasingly, and consequently there are $v$, $u''$ and $v'$ such that

$$t' \underset{\gamma\Gamma}{\overset{*}{\longrightarrow}} \cdot \underset{\Delta}{\shortparallel\!\!\!\!\!\!\rightarrow} \cdot \underset{\gamma\Gamma\Delta}{\overset{*}{\longrightarrow}} v' \underset{\gamma\Gamma\Delta}{\overset{*}{\longleftarrow}} v \overset{Q'}{\underset{\Gamma}{\leftarrow\!\!\!\!\!\!\shortparallel}} u'' \underset{\gamma\Delta}{\overset{*}{\longleftarrow}} u'$$

with $\mathcal{V}\mathsf{ar}(v|_{Q'}) \subseteq \mathcal{V}\mathsf{ar}(s|_{P'})$. Consequently,

$$t = t'\sigma' \underset{\gamma\Gamma}{\overset{*}{\longrightarrow}} \cdot \underset{\Delta}{\shortparallel\!\!\!\!\!\!\rightarrow} \cdot \underset{\gamma\Gamma\Delta}{\overset{*}{\longrightarrow}} v'\sigma' \underset{\gamma\Gamma\Delta}{\overset{*}{\longleftarrow}} v\sigma' \underset{\Gamma}{\leftarrow\!\!\!\!\!\!\shortparallel} v\sigma \overset{Q'}{\underset{\Gamma}{\leftarrow\!\!\!\!\!\!\shortparallel}} u'' \underset{\gamma\Delta}{\overset{*}{\longleftarrow}} u'\sigma = u$$

Since $\sigma(x) = \sigma'(x)$ for $x \in \mathcal{V}\mathsf{ar}(s|_{P'})$ (otherwise $s \underset{\Gamma}{\shortparallel\!\!\!\!\!\!\rightarrow} t$ would not be a parallel step), and because $\mathcal{V}\mathsf{ar}(v|_{Q'}) \subseteq \mathcal{V}\mathsf{ar}(s|_{P'})$, the two parallel steps in the leftward sequence can be combined into a single one. Thus we can join $t$ and $u$ decreasingly with common reduct $v'\sigma'$, completing the proof. See also Figure 1. □

To conclude the section we demonstrate Theorem 4 on two examples.

*Example* 1. Consider the TRS $\mathcal{R}$ consisting of the following five rules with labels $2 \succ 1 \succ 0$:

$$\mathsf{a} \underset{1}{\to} \mathsf{b} \qquad \mathsf{b} \underset{0}{\to} \mathsf{a} \qquad \mathsf{f}(\mathsf{a},\mathsf{a}) \underset{1}{\to} \mathsf{c} \qquad \mathsf{f}(\mathsf{b},\mathsf{b}) \underset{2}{\to} \mathsf{c} \qquad \mathsf{h}(x) \underset{0}{\to} \mathsf{h}(\mathsf{f}(x,x))$$

There are six parallel critical pairs that can all be joined decreasingly as required by Theorem 4:

$$\mathsf{f}(\{\mathsf{a},\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}) \underset{\{1\}}{\leftarrow\!\!\shortparallel} \mathsf{f}(\mathsf{a},\mathsf{a}) \underset{\{1\}}{\longrightarrow} \mathsf{c} : \qquad \mathsf{f}(\{\mathsf{a},\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}) \underset{\{0\}}{\shortparallel\!\!\rightarrow} \mathsf{f}(\mathsf{a},\mathsf{a}) \underset{\{1\}}{\longrightarrow} \mathsf{c}$$

$$\mathsf{f}(\{\mathsf{a},\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}) \underset{\{0\}}{\leftarrow\!\!\shortparallel} \mathsf{f}(\mathsf{b},\mathsf{b}) \underset{\{2\}}{\longrightarrow} \mathsf{c} : \qquad \mathsf{f}(\{\mathsf{a},\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}) \underset{\{0\}}{\shortparallel\!\!\rightarrow} \mathsf{f}(\mathsf{a},\mathsf{a}) \underset{\{1\}}{\longrightarrow} \mathsf{c}$$

Therefore, $\mathcal{R}$ is confluent.

*Example* 2. Let $\mathcal{O}$ be the TRS consisting of the rules (confluence of $\mathcal{O}$ was shown in [4])

$$\begin{array}{rrrr}
x - 0 \underset{0}{\to} x & 0 - x \underset{0}{\to} 0 & \mathsf{s}(x) - \mathsf{s}(y) \underset{0}{\to} x - y & \mathsf{if}(\mathsf{true}, x, y) \underset{0}{\to} x \\
0 < \mathsf{s}(x) \underset{0}{\to} \mathsf{true} & x < 0 \underset{0}{\to} \mathsf{false} & \mathsf{s}(x) < \mathsf{s}(y) \underset{0}{\to} x < y & \mathsf{if}(\mathsf{false}, x, y) \underset{0}{\to} y \\
\mathsf{mod}(x, 0) \underset{0}{\to} x & \mathsf{mod}(0, x) \underset{0}{\to} 0 & \multicolumn{2}{c}{\mathsf{mod}(x, \mathsf{s}(y)) \underset{1}{\to} \mathsf{if}(x < \mathsf{s}(y), x, \mathsf{mod}(x - \mathsf{s}(y), \mathsf{s}(y)))} \\
\mathsf{gcd}(x, 0) \underset{0}{\to} x & \mathsf{gcd}(0, x) \underset{0}{\to} x & \multicolumn{2}{c}{\mathsf{gcd}(x, y) \underset{1}{\to} \mathsf{gcd}(y, \mathsf{mod}(x, y))}
\end{array}$$

There are 12 critical pairs, of which 6 are trivial, and the remaining 6 come in 3 symmetrical pairs. They can all be joined decreasingly, using the precedence $1 \succ 0$:

$$\mathsf{if}(0 < \mathsf{s}(\mathsf{y}), 0, \mathsf{mod}(0 - \mathsf{s}(y), \mathsf{s}(y))) \underset{\{1\}}{\leftarrow\!\!\shortparallel} \mathsf{mod}(0, \mathsf{s}(y)) \underset{\{0\}}{\longrightarrow} 0 : \quad \ldots \underset{\{0\}}{\longrightarrow} \mathsf{if}(\mathsf{true}, 0, \ldots) \underset{\{0\}}{\longrightarrow} 0$$

$$\mathsf{gcd}(0, \mathsf{mod}(x, 0)) \underset{\{1\}}{\leftarrow\!\!\shortparallel} \mathsf{gcd}(x, 0) \underset{\{0\}}{\longrightarrow} x : \quad \ldots \underset{\{0\}}{\longrightarrow} \mathsf{gcd}(x, 0) \underset{\{0\}}{\longrightarrow} x$$

$$\mathsf{gcd}(x, \mathsf{mod}(0, x)) \underset{\{1\}}{\leftarrow\!\!\shortparallel} \mathsf{gcd}(0, x) \underset{\{0\}}{\longrightarrow} x : \quad \ldots \underset{\{0\}}{\longrightarrow} \mathsf{gcd}(0, x) \underset{\{0\}}{\longrightarrow} x$$

There are no inner critical pairs, so this accounts for all parallel critical pairs, and we can conclude that $\mathcal{O}$ is confluent by Theorem 4.

# 4 Generalized Labelings

We implemented rule-labeling for left-linear TRSs in the confluence tool CSI [8]. However, the implementation does not use Theorem 4. Instead, we use the framework of labelings from [9], in particular weak ll-labelings, which we recall below. Let $\mathcal{L}$ be a set of labels equipped with a well-founded order $\succ$, and a quasi-order $\succeq$ on $\mathcal{L}$ that is compatible with $\succ$, i.e., $\succeq \cdot \succ \cdot \succeq \subseteq \succ$.[1]

**Definition 5.** A *labeling* is a function $\ell$ mapping rewrite steps to labels such that comparing labels is closed under contexts and substitutions, that is, for all contexts $C[\cdot]$, substitutions $\sigma$ and rewrite steps $s \to t$, $s' \to t'$, we have

$$\ell(C[s\sigma] \to C[t\sigma]) \succ \ell(C[s'\sigma] \to C[t'\sigma]) \quad \text{if} \quad \ell(s \to t) \succ \ell(s' \to t')$$
$$\ell(C[s\sigma] \to C[t\sigma]) \succeq \ell(C[s'\sigma] \to C[t'\sigma]) \quad \text{if} \quad \ell(s \to t) \succeq \ell(s' \to t')$$

A labeling is a *weak ll-labeling* if for all $s \xrightarrow{p} t$ and $s \xrightarrow{p'} t'$ with $p \parallel p'$ (parallel overlap), we have

$$\ell(s \to t) \succeq \ell(t \to t'[t|_p]_p) \tag{2}$$

and whenever $s \xrightarrow{\epsilon}_{l \to r} t$, $s \xrightarrow{pq} t'$ with $p \in \mathcal{P}\mathsf{os}_{\mathcal{V}}(l)$ (variable overlap), if $l|_p = r|_{p'}$, we have[2]

$$\ell(s \to t') \succeq \ell(t \to t[t'|_p]_{p'}) \tag{3}$$

[1] Another common definition is $\succ \cdot \succeq \subseteq \succ$, in which case $\succ' = \succeq \cdot \succ$ satisfies $\succeq \cdot \succ' \cdot \succeq \subseteq \succ'$.

[2] Condition (3) is stronger than the left-linear variable overlap condition in [9], where the parallel step of the joining valley is a rewrite sequence, but the weak ll-labelings presented there satisfy our new condition as well.

Given a weak ll-labeling $\ell$, we can define $\ell^{\parallel}$ as before in Definition 2. However, it appears that we cannot use the multiset extension of $(\succ, \succeq)$ for comparing labels. Instead, we use the Hoare order on sets,[3]

$$\Gamma \succ \Delta \iff \Gamma \neq \varnothing \wedge \forall \beta \in \Delta\, \exists \alpha \in \Gamma\, (\alpha \succ \beta) \quad \text{and} \quad \Gamma \succeq \Delta \iff \forall \beta \in \Delta\, \exists \alpha \in \Gamma\, (\alpha \succeq \beta)$$

The implementation is based on the following theorem.

**Theorem 6.** *A left-linear TRS $\mathcal{R}$ is confluent if all its parallel critical peaks $t \overset{P}{\underset{\Gamma}{\leftarrow\!\!\!\!\!\!\parallel}} s \underset{\Delta}{\longrightarrow\!\!\!\!\!\!\parallel} u$ can be joined decreasingly (with respect to some ll-labeling) as $t \xrightarrow[\curlyvee\Gamma]{*} \cdot \underset{\Delta'}{\longrightarrow\!\!\!\!\!\!\parallel} \cdot \xrightarrow[\curlyvee\Gamma\Delta]{*} \cdot \xleftarrow[\curlyvee\Gamma\Delta]{*} v \overset{Q}{\underset{\Gamma'}{\leftarrow\!\!\!\!\!\!\parallel}} \cdot \xleftarrow[\curlyvee\Delta]{*} u$ such that $\mathcal{V}\mathsf{ar}(v|_Q) \subseteq \mathcal{V}\mathsf{ar}(s|_P)$ and $\Delta \succeq \Delta'$, $\Gamma \succeq \Gamma'$.*

The proof of Theorem 6 is similar to that of Theorem 4, but note that we have to consider all parallel critical pairs now; there is no analogue of Proposition 3. In order to find suitable labelings, we use the techniques of [9] on joining sequences for critical pairs, and finally check whether the resulting labeling allows joining parallel critical pairs decreasingly as well.

# 5 Conclusion

We have demonstrated a rule labeling technique based on parallel reductions and parallel critical pairs that works for general left-linear term rewrite systems. A variant of the approach is implemented in CSI, which found the proofs for Examples 1,2. As future work we plan to flesh out the generalization from Section 4, explore the different orders on sets. There is also room for better heuristics for finding suitable ll-labelings. We would also like to investigate whether the variable condition $\mathcal{V}\mathsf{ar}(v|_Q) \subseteq \mathcal{V}\mathsf{ar}(s|_P)$ in Theorems 4, 6 can be relaxed.

It would also be interesting to devise a conversion version of Theorem 6. This should be straightforward in the pure rule labeling setting (Theorem 4), but it is unclear whether the generalization to weak ll-labelings would work out.

# References

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

[2] B. Gramlich. Confluence without termination via parallel critical pairs. In *Proc. 21st CAAP*, volume 1059 of *LNCS*, pages 211–225, 1996.

[3] N. Hirokawa and A. Middeldorp. Decreasing diagrams and relative termination. In *Proc. 5th IJCAR*, volume 6173 of *LNCS (LNAI)*, pages 487–501, 2010.

[4] M. Oyamaguchi and Y. Ohta. On the Church-Rosser property of left-linear term rewriting systems. *IEICE TIS*, E86-D(1):131–135, 2003.

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

[6] V. van Oostrom. Confluence by decreasing diagrams. *TCS*, 126(2):259–280, 1994.

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

[8] H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – A confluence tool. In *Proc. 23rd CADE*, volume 6803 of *LNCS (LNAI)*, pages 499–505, 2011.

[9] H. Zankl, B. Felgenhauer, and A. Middeldorp. Labelings for decreasing diagrams. In *Proc. 22nd RTA*, volume 10 of *LIPIcs*, pages 377–392, 2011.

[3] This order was introduced for power domains in domain theory.

# Commutation via Relative Termination*

Nao Hirokawa
JAIST, Japan
hirokawa@jaist.ac.jp

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

**Abstract**

We present a generalisation of a commutation criterion by Rosen (1973), the development closedness theorem by van Oostrom (1994), and a confluence criterion by Hirokawa and Middeldorp (2011).

## 1 Introduction

This note is about sufficient conditions for the *commutation* of two *left-linear* TRSs $\mathcal{R}$ and $\mathcal{S}$. Commutation means that the inclusion

$$\xleftarrow[\mathcal{R}]{*} \cdot \xrightarrow[\mathcal{S}]{*} \;\subseteq\; \xrightarrow[\mathcal{S}]{*} \cdot \xleftarrow[\mathcal{R}]{*}$$

holds. When $\mathcal{R} = \mathcal{S}$, commutation amounts to confluence. Furthermore, commutation is a sufficient condition for the confluence of the union of two confluent TRSs. It is well-known that $\mathcal{R}$ and $\mathcal{S}$ commute when there are no critical pairs between the rules of $\mathcal{R}$ and the rules of $\mathcal{S}$. Toyama [6] generalized this result of Rosen [4] (and Raoult and Vuillemin [3, Proposition 10]) by admitting critical pairs between $\mathcal{R}$ and $\mathcal{S}$ but restricting the way in which they are joined. More precisely, Toyama showed that $\mathcal{R}$ and $\mathcal{S}$ commute if the following two conditions are satisfied:

$$\xleftarrow[\mathcal{R}]{} \rtimes \xrightarrow[\mathcal{S}]{} \;\subseteq\; \xrightarrow[\mathcal{S}]{\shortparallel} \cdot \xleftarrow[\mathcal{R}]{*} \qquad \text{and} \qquad \xleftarrow[\mathcal{S}]{>\epsilon} \rtimes \xrightarrow[\mathcal{R}]{} \;\subseteq\; \xrightarrow[\mathcal{R}]{\shortparallel}$$

As observed in [7, Corollaries 24 and 28] and [1, Proposition 7], Toyama's result is further generalized by adopting multi-steps:

$$\xleftarrow[\mathcal{R}]{} \rtimes \xrightarrow[\mathcal{S}]{} \;\subseteq\; \xrightarrow[\mathcal{S}]{\circ} \cdot \xleftarrow[\mathcal{R}]{*} \qquad \text{and} \qquad \xleftarrow[\mathcal{S}]{>\epsilon} \rtimes \xrightarrow[\mathcal{R}]{} \;\subseteq\; \xrightarrow[\mathcal{R}]{\circ}$$

We present a generalisation of these results by incorporating the confluence result based on relative termination [2].

## 2 Preliminaries

We assume familiarity with the conversion version of the decreasing diagrams technique for proving commutation, due to van Oostrom [8, Theorem 3]. Familiarity with proof terms witnessing multi-steps ([5, Chapter 8], [2]) is also helpful. Below we recall some basic definitions and recast a measure used in [7] as a measure on co-initial proof terms.

Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. For each rule $\ell \to r \in \mathcal{R}$ we introduce a fresh *rule symbol* $\underline{\ell \to r}$ whose arity is given by the number of variables in $\ell$. *Proof terms* are terms over functions in $\mathcal{F}$ and rule symbols. We write $\sqsubseteq$ for the smallest rewrite preorder on proof terms such that $\ell \sqsubseteq \underline{\ell \to r}(x_1, \ldots, x_n)$ for all rules $\ell \to r \in \mathcal{R}$ with $\mathsf{var}(\ell) = (x_1, \ldots, x_n)$. Here $\mathsf{var}(\ell)$ denotes a sequence consisting of all variables in $\mathcal{V}\mathsf{ar}(\ell)$ in some fixed order. Given co-initial

*Supported by the JSPS KAKENHI Grant Number 25730004 and the Austrian Science Fund (FWF) P22467.

proof terms $A$ and $B$, the (partial) residual operation is the proof term $A \setminus B$ defined by the following clauses (here $\langle A_1, \ldots, A_n \rangle_\ell$ stands for the substitution $\{x_i \mapsto A_i \mid 1 \leqslant i \leqslant n\}$):

$$\begin{aligned}
x \setminus x &= x \\
f(A_1, \ldots, A_n) \setminus f(B_1, \ldots, B_n) &= f(A_1 \setminus B_1, \ldots, A_n \setminus B_n) \\
\underline{\ell \to r}(A_1, \ldots, A_n) \setminus \underline{\ell \to r}(B_1, \ldots, B_n) &= r\langle A_1 \setminus B_1, \ldots, A_n \setminus B_n \rangle_\ell \\
\underline{\ell \to r}(A_1, \ldots, A_n) \setminus \ell\langle B_1, \ldots, B_n \rangle_\ell &= \underline{\ell \to r}(A_1 \setminus B_1, \ldots, A_n \setminus B_n) \\
\ell\langle A_1, \ldots, A_n \rangle_\ell \setminus \underline{\ell \to r}(B_1, \ldots, B_n) &= r\langle A_1 \setminus B_1, \ldots, A_n \setminus B_n \rangle_\ell
\end{aligned}$$

When we use $A \setminus B$ below, it is guaranteed to be defined. (If $A \sqsubseteq B$ then $A \setminus B$ is a proof term without rule symbols.) A *redex* is a proof term that contains exactly one rule symbol. For a redex $\Delta$ we write $\Delta \in A$ if $\Delta \sqsubseteq A$. Given a proof term $A$ and a redex $\Delta \in A$, we find it convenient to write $A - \Delta$ for the proof term that is obtained from $A$ by replacing the subterm occurrence $\underline{\ell \to r}(A_1, \ldots, A_n)$ corresponding to $\Delta$ by $\ell\langle A_1, \ldots, A_n \rangle_\ell$.

For redexes $\Delta_1, \Delta_2 \in A$ we write $\Delta_1 \geqslant \Delta_2$ if the rule symbol of $\Delta_1$ appears at or below that of $\Delta_2$ in $A$. The set $\blacktriangle(A)$ of positions is inductively defined on $A$ as follows:

$$\begin{aligned}
\blacktriangle(x) &= \varnothing \\
\blacktriangle(f(A_1, \ldots, A_n)) &= \{iq \mid 1 \leqslant i \leqslant n \text{ and } q \in \blacktriangle(A_i)\} \\
\blacktriangle(\underline{\ell \to r}(A_1, \ldots, A_n)) &= \mathcal{P}\mathsf{os}_{\mathcal{F}}(\ell) \cup \{pq \mid p \in \mathcal{P}\mathsf{os}_{\{x_i\}}(\ell) \text{ and } q \in \blacktriangle(A_i)\}
\end{aligned}$$

where $x \in \mathcal{V}$, $f \in \mathcal{F}$, and $\mathsf{var}(\ell) = (x_1, \ldots, x_n)$. So $\blacktriangle(A)$ consists of all function positions of the redex patterns contracted in the multi-step of $A$. We write $\blacktriangle(A, B)$ for $\blacktriangle(A) \cap \blacktriangle(B)$. The set $\blacktriangle(A, B)$ is used below as a measure for the amount of overlap between the proof terms $A$ and $B$. A pair $(\Delta_1, \Delta_2)$ of co-initial redexes is an *overlap* if $\blacktriangle(\Delta_1, \Delta_2) \neq \varnothing$.

**Lemma 2.1.** *Let $A$ and $B$ be proof terms of left-linear TRSs $\mathcal{R}$ and $\mathcal{S}$ respectively. If $\blacktriangle(A, B) \neq \varnothing$ then there exist a trivial or critical peak $t \mathrel{{}^{\Delta_1}_{\mathcal{R}}\!\!\leftarrow} s \mathrel{\rightarrow\!\!{}^{\Delta_2}_{\mathcal{S}}} u$, a context $\mathbb{C}$, and a substitution $\sigma$ such that $\mathbb{C}[\Delta_1\sigma] \in A$ and $\mathbb{C}[\Delta_2\sigma] \in B$. If $\blacktriangle(A, B) = \varnothing$ then $\mathrel{{}^{A}_{\mathcal{R}}\!\!\leftarrow} \cdot \mathrel{\rightarrow\!\!{}^{B}_{\mathcal{S}}} \subseteq \mathrel{\rightarrow\!\!{}^{B \setminus A}_{\mathcal{S}}} \cdot \mathrel{{}^{A \setminus B}_{\mathcal{R}}\!\!\leftarrow}$.* □

# 3 Commutation by Relative Termination

Let $\mathcal{R}$ and $\mathcal{S}$ be TRSs. The set of critical peak steps of $\mathcal{S}$ for $\mathcal{R}$ is defined as follows:

$$\mathsf{CPS}_{\mathcal{R}}(\mathcal{S}) = \{s \to u \mid t \mathrel{{}_{\mathcal{R}}\!\!\leftarrow} s \to_{\mathcal{S}} u \text{ is a critical peak}\}$$

**Lemma 3.1.** *Let $\mathcal{R}$ and $\mathcal{S}$ be left-linear TRSs. If $t \mathrel{{}_{\mathcal{R}}\!\!\leftarrow\!\!\circ} s \mathrel{\circ\!\!\rightarrow_{\mathcal{S}}} u$ then $t \mathrel{\circ\!\!\rightarrow_{\mathcal{S}}} \cdot \mathrel{{}_{\mathcal{R}}\!\!\leftarrow\!\!\circ} u$ or $t \mathrel{{}_{\mathcal{R}}\!\!\leftarrow\!\!\circ} \cdot {}_{\mathsf{CPS}_{\mathcal{S}}(\mathcal{R})}\!\!\leftarrow s \to_{\mathsf{CPS}_{\mathcal{R}}(\mathcal{S})} \cdot \mathrel{\circ\!\!\rightarrow_{\mathcal{S}}} u$.* □

**Theorem 3.2.** *Left-linear locally commuting TRSs $\mathcal{R}$ and $\mathcal{S}$ commute if $\mathsf{CPS}_{\mathcal{S}}(\mathcal{R}) \cup \mathsf{CPS}_{\mathcal{R}}(\mathcal{S})$ is relatively terminating over $\mathcal{R} \cup \mathcal{S}$.*

*Proof.* We use decreasing diagrams with the predecessor labeling. To this end we define $t \mathrel{\circ\!\!\rightarrow_{\mathcal{R},s}} u$ if and only if $s \to^*_{\mathcal{R}\cup\mathcal{S}} t \mathrel{\circ\!\!\rightarrow_{\mathcal{R}}} u$. The labeled relation $\mathrel{\circ\!\!\rightarrow_{\mathcal{S},s}}$ is defined similarly. We write $>$ to denote $\to^+_{(\mathsf{CPS}_{\mathcal{R}}(\mathcal{S})\cup\mathsf{CPS}_{\mathcal{S}}(\mathcal{R}))/(\mathcal{R}\cup\mathcal{S})}$. By the relative termination condition, $>$ is a well-founded order. We show that $(\{\mathrel{\circ\!\!\rightarrow_{\mathcal{R},s}}\}_{s\in\mathcal{T}}, \{\mathrel{\circ\!\!\rightarrow_{\mathcal{S},s}}\}_{s\in\mathcal{T}})$ is decreasing with respect to $>$. Suppose $t \mathrel{{}_{\mathcal{R},s_1}\!\!\leftarrow\!\!\circ} s \mathrel{\circ\!\!\rightarrow_{\mathcal{S},s_2}} u$. By Lemma 3.1 and local commutation there are terms $v$ and $w$ such that (a) or (b) of Figure 1 holds. Note that $s_i > v, w$ holds in case (b) because $s_i \to^*_{\mathcal{R}\cup\mathcal{S}} s > v, w$. Therefore, in both cases decreasingness is established. Hence $\mathcal{R}$ and $\mathcal{S}$ commute. □

Theorem 3.2 subsumes Rosen's commutativity criterion [4], as mutual orthogonality implies $\mathsf{CPS}_{\mathcal{S}}(\mathcal{R}) \cup \mathsf{CPS}_{\mathcal{R}}(\mathcal{S}) = \varnothing$ as well as local commutation.

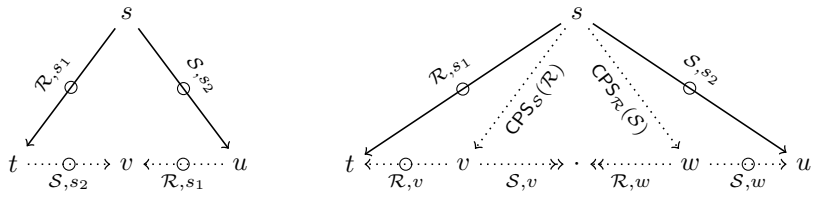



Figure 1: Decreasingness of $(\multimap_{\mathcal{R}}, \multimap_{\mathcal{S}})$.

# 4 Incorporating Development Closedness

We extend the last theorem in order to subsume the development closedness theorem of van Oostrom [7]. Let $t \mathrel{\mathcal{R}}\!\leftarrow^{p} s \xrightarrow{q}_{\mathcal{S}} u$ be a critical peak. By definition $p$ or $q$ must be the root position. We say that the peak is $(\mathcal{R},\mathcal{S})$-*closed* if $t \mathrel{\mathcal{R}}\!\leftarrow\!\multimap u$ whenever $p = \epsilon$ and $t \multimap\!\!\to_{\mathcal{S}} u$ whenever $q = \epsilon$. The set of all non-closed critical peak steps of $\mathcal{S}$ for $\mathcal{R}$ is defined as follows:

$$\mathsf{CPS}'_{\mathcal{R}}(\mathcal{S}) = \{s \to u \mid t \mathrel{\mathcal{R}}\!\leftarrow s \to_{\mathcal{S}} u \text{ is a critical peak which is not } (\mathcal{R},\mathcal{S})\text{-closed}\}$$

**Lemma 4.1.** *Let $\mathcal{R}$ and $\mathcal{S}$ be left-linear TRSs. If $t \mathrel{\mathcal{R}}\!\leftarrow\!\multimap s \multimap\!\!\to_{\mathcal{S}} u$ then (a) $t \multimap\!\!\to_{\mathcal{S}} \cdot \mathrel{\mathcal{R}}\!\leftarrow\!\multimap u$ or (b) $t \mathrel{\mathcal{R}}\!\leftarrow\!\multimap \cdot \mathrel{\mathsf{CPS}'_{\mathcal{S}}(\mathcal{R})}\!\leftarrow s' \to_{\mathsf{CPS}'_{\mathcal{R}}(\mathcal{S})} \cdot \multimap\!\!\to_{\mathcal{S}} u$ and $s \to^*_{\mathcal{R}\cup\mathcal{S}} s'$ for some $s'$.*

Lemma 4.1 is a proper generalization of [2, Lemma 4]. In order to prove the lemma we introduce some more notions. Whenever $\blacktriangle(A,B)$ is non-empty, there exists an innermost overlap. An overlap $(\Delta_1,\Delta_2)$ in $A \times B$ is called *innermost* if there are no other overlaps $(\Delta'_1,\Delta'_2)$ in $A \times B$ such that $\Delta'_1 \geqslant \Delta_1$ and $\Delta'_2 \geqslant \Delta_2$. We say that $(\Delta_1,\Delta_2)$ is $(\mathcal{R},\mathcal{S})$-closed if the corresponding critical peak is $(\mathcal{R},\mathcal{S})$-closed. The following is the key lemma. In order to pave the way for formalisation of advanced confluence and commutation results, we formalise the proof technique introduced in [7] using proof terms.

**Lemma 4.2.** *Suppose $(\Delta_1,\Delta_2)$ is an innermost overlap in $A \times B$ with $\Delta_1 \geqslant \Delta_2$. If $(\Delta_1,\Delta_2)$ is $(\mathcal{R},\mathcal{S})$-closed then $\blacktriangle(A,B) \supsetneq \blacktriangle(A \setminus \Delta_1, C)$ for some proof term $C$ that is co-initial with $A \setminus \Delta_1$.*

*Proof.* Write $t \xleftarrow[\mathcal{R}]{A \setminus \Delta_1} t_1 \xleftarrow[\mathcal{R}]{\Delta_1} s \xrightarrow[\mathcal{S}]{\Delta_2} u_1 \xrightarrow[\mathcal{S}]{B \setminus \Delta_2} u$ and let

$$t'_1 \xleftarrow[\mathcal{R}]{\Delta'_1} s' \xrightarrow[\mathcal{S}]{\Delta'_2} u'_1$$

be the corresponding critical peak. Since $s \trianglerighteq s'$, there exist a context $\mathbb{C}$ and a substitution $\sigma$ such that $s = \mathbb{C}[s'\sigma]$, $t_1 = \mathbb{C}[t'_1\sigma]$, $u_1 = \mathbb{C}[u'_1\sigma]$, $\Delta_1 = \mathbb{C}[\Delta'_1\sigma]$, and $\Delta_2 = \mathbb{C}[\Delta'_2\sigma]$. We have $t'_1 \multimap\!\!\to_{\mathcal{R}} u'_1$ because the critical peak is $(\mathcal{R},\mathcal{S})$-closed. Let $D'$ be the proof term witnessing this multi-step and define $D = \mathbb{C}[D'\sigma]$. Clearly, $D$ is a witness of $t_1 \multimap\!\!\to_{\mathcal{R}} u_1$. We will compose $D$ and $B \setminus \Delta_2$ into a single proof term $C\colon t_1 \multimap\!\!\to_{\mathcal{R}} u$. Let $B' = B - \Delta_2$. Since $(\Delta_1,\Delta_2)$ is an innermost overlap with $\Delta_1 \geqslant \Delta_2$, $\blacktriangle(B',\Delta_1) = \varnothing$. Hence $B' \setminus \Delta_1$ is well-defined. If we show that $\blacktriangle(B' \setminus \Delta_1, D) = \varnothing$ then we can define $C$ as $(B' \setminus \Delta_1) \sqcup D$. We have $\blacktriangle(D') \subseteq \mathcal{P}\mathsf{os}_{\mathcal{F}}(t'_1)$ and thus $\blacktriangle(D) \subseteq \{qp \mid p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(t'_1)\}$ where $q$ is the position of the hole in $\mathbb{C}$. In words, $D$ affects only the $t'_1$ part of $\mathbb{C}[t'_1\sigma]$. Let $\Delta \in B'$ be an arbitrary redex. We show that $\blacktriangle(\Delta \setminus \Delta_1, D) = \varnothing$. We split $B'$ into two parts: $B' = B_1 \sqcup B_2$ with

$$B_1 = B' - B_2 \qquad\qquad B_2 = \bigsqcup\{\Delta \mid \Delta \in B' \text{ and } \Delta > \Delta_2\}$$

Depending on the position of $\Delta$, we distinguish two cases:

(a) If $\Delta \in B_1$ then $\Delta$ does not overlap $\Delta_1$. So $\blacktriangle(\Delta \setminus \Delta_1)$ consists entirely of positions in $\mathbb{C}$ and thus $\blacktriangle(\Delta \setminus \Delta_1, D) = \varnothing$.

(b) Otherwise, $\Delta \in B_2$. Since $(\Delta_1, \Delta_2)$ is an innermost overlap, $\blacktriangle(\Delta, \Delta_1) = \varnothing$. Because $\Delta \neq \Delta_2$ we also have $\blacktriangle(\Delta, \Delta_2) = \varnothing$. The crucial observation is that no redex in $\Delta \setminus \Delta_1$ overlaps with $D$. This follows because for the originating term $s'$ of the critical peak we have $\mathcal{P}\mathsf{os}_{\mathcal{F}}(s') = \blacktriangle(\Delta_1') \cup \blacktriangle(\Delta_2')$, due to the fact that the involved rules are left-linear. Hence all redexes in $\Delta \setminus \Delta_1$ appear in the substitution part $\sigma$ of $\mathbb{C}[t_1'\sigma]$.

We conclude that $\blacktriangle(B' \setminus \Delta_1, D) = \varnothing$. It remains to show that $\blacktriangle(A, B) \supsetneq \blacktriangle(A \setminus \Delta_1, C)$. We have $\blacktriangle(B_1 \setminus \Delta_1) = \blacktriangle(B_1)$ and hence $\blacktriangle(C) = \blacktriangle(B_1) \cup \blacktriangle(B_2 \setminus \Delta_1) \cup \blacktriangle(D)$. Because $(\Delta_1, \Delta_2)$ is an innermost overlap, $\blacktriangle(A, B_2) = \varnothing$ and therefore

$$\blacktriangle(A, B) = \blacktriangle(A, \Delta_2) \cup \blacktriangle(A, B_1)$$

and $\blacktriangle(A \setminus \Delta_1, B_2 \setminus \Delta_1) = \varnothing$. Because all redexes in $B_1$ occur above $\Delta_2$, they do not overlap with $\Delta_1$ and thus $\blacktriangle(A \setminus \Delta_1, B \setminus \Delta_1) = \blacktriangle(A \setminus \Delta_1, B_1 \setminus \Delta_1) = \blacktriangle(A, B_1)$. We obtain

$$\begin{aligned}\blacktriangle(A \setminus \Delta_1, C) &= \blacktriangle(A \setminus \Delta_1, B' \setminus \Delta_1) \cup \blacktriangle(A \setminus \Delta_1, D)\\ &= \blacktriangle(A, B_1) \cup \blacktriangle(A \setminus \Delta_1, D)\end{aligned}$$

and so it remains to show that $\blacktriangle(A, \Delta_2) \supsetneq \blacktriangle(A \setminus \Delta_1, D)$. We split $A$ into three parts: $A = A_1 \sqcup \Delta_1 \sqcup A_2$ with $A_1 = A - \Delta_1 - A_2$ and $A_2 = \bigsqcup\{\Delta \mid \Delta \in A \text{ and } \Delta > \Delta_1\}$. We have $\blacktriangle(A, \Delta_2) = \blacktriangle(A_1, \Delta_2) \cup \blacktriangle(\Delta_1, \Delta_2)$ and

$$\blacktriangle(A \setminus \Delta_1, D) = \blacktriangle(A_1 \setminus \Delta_1, D) \cup \blacktriangle(\Delta_1 \setminus \Delta_1, D) \cup \blacktriangle(A_2 \setminus \Delta_1, D)$$

The second component is clearly empty but also the third component reduces to the empty set, by repeating the reasoning in case (b) above. The first component equals $\blacktriangle(A_1, D)$ since redexes in $A_1$ occur above $\Delta_1$. So $\blacktriangle(A \setminus \Delta_1, D) = \blacktriangle(A_1, D)$. Since $\blacktriangle(\Delta_1, \Delta_2) \neq \varnothing$, it suffices to show the inclusion $\blacktriangle(A_1, D) \subseteq \blacktriangle(A_1, \Delta_2)$, which follows from the observation that overlaps between $A_1$ and $D$ are restricted to positions in $\blacktriangle(\Delta_2) - \blacktriangle(\Delta_1)$. □

We are ready to show the aforementioned lemma.

*Proof of Lemma 4.1.* By induction on the finite set $\blacktriangle(A, B)$ with respect to the well-founded order $\supsetneq$. Let

$$t \xleftarrow[\mathcal{R}]{A} s \xrightarrow[\mathcal{S}]{B} u$$

If $\blacktriangle(A, B)$ is empty then (a) holds by the second part of Lemma 2.1. Otherwise, there is an overlap in $A \times B$. We distinguish two cases.

- If a non-closed overlap $(\Delta_1, \Delta_2)$ exists in $A \times B$, (b) holds because

$$t \xleftarrow[\mathcal{R}]{A \setminus \Delta_1} \cdot \xleftarrow[\mathcal{R}]{\Delta_1} s \xrightarrow[\mathcal{S}]{\Delta_2} \cdot \xrightarrow[\mathcal{S}]{B \setminus \Delta_2} u$$

  and the two inner steps correspond to applications of rules from $\mathsf{CPS}'_{\mathcal{S}}(\mathcal{R})$ and $\mathsf{CPS}'_{\mathcal{R}}(\mathcal{S})$.

- In the other case all overlaps are closed. Let $(\Delta_1, \Delta_2)$ be an innermost overlap. Without loss of generality we assume $\Delta_1 \geqslant \Delta_2$. According to Lemma 4.2 there exists some proof

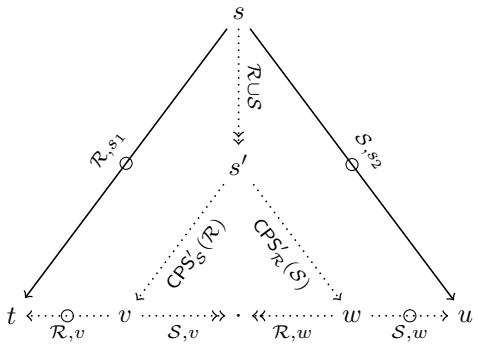
term $C$ such that $\blacktriangle(A, B) \supsetneq \blacktriangle(A \setminus \Delta_1, C)$. Let $s'$ be the term such that $s \xrightarrow{\Delta_1} s'$. (In the proof of Lemma 4.2 $s'$ was called $t_1$.) Because

$$t \xleftarrow[\mathcal{R}]{A \setminus \Delta_1} s' \xrightarrow[\mathcal{S}]{C} u$$

the induction hypothesis applies. □

**Theorem 4.3.** *Left-linear locally commuting TRSs $\mathcal{R}$ and $\mathcal{S}$ commute if $\mathsf{CPS}'_{\mathcal{S}}(\mathcal{R}) \cup \mathsf{CPS}'_{\mathcal{R}}(\mathcal{S})$ is relatively terminating over $\mathcal{R} \cup \mathcal{S}$.*

*Proof.* Use Lemma 4.1 to replace Figure 1(b) by the following diagram:




□

Due to lack of space, we omit the extension that weakens the joinability requirement for overlays, but note that the result of Theorem 4.3 remains true if we strengthen the notion of $(\mathcal{R}, \mathcal{S})$-closedness by allowing $t \multimap^*_{\mathcal{S}} \cdot {}_{\mathcal{R}}\!\multimap u$ when $p = \epsilon$. Future work includes the extension to critical valleys [9] as well as automation. Experimental data on the confluence problem database Cops[1] for the presented theorems will be reported at the workshop.

# References

[1] T. Aoto, J. Yoshida, and Y. Toyama. Proving confluence of term rewriting systems automatically. In *Proc. 21st RTA*, volume 5595 of *LNCS*, pages 93–102, 2009.

[2] N. Hirokawa and A. Middeldorp. Decreasing diagrams and relative termination. *Journal of Automated Reasoning*, 47(4):481–501, 2011.

[3] J.-C. Raoult and J. Vuillemin. Operational and semantic equivalence between recursive programs. *Journal of the ACM*, 27(4):772–796, 1980.

[4] B.K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, 1973.

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

[6] Y. Toyama. Commutativity of term rewriting systems. In *Programming of Future Generation Computers II*, pages 393–407. North-Holland, 1988.

[7] V. van Oostrom. Developing developments. *Theoretical Computer Science*, 175(1):159–181, 1997.

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

[9] V. van Oostrom. Confluence via critical valleys. In *Proc. 6th HOR*, pages 9–10, 2012.

[1] `http://coco.nue.riec.tohoku.ac.jp/`

# Proving Confluence of Conditional Term Rewriting Systems via Unravelings *

Karl Gmeiner[1], Naoki Nishida[2] and Bernhard Gramlich[1]

[1] Faculty of Informatics, TU Wien, Austria
{gmeiner, gramlich}@logic.at
[2] Graduate School of Information Science, Nagoya University, Japan
nishida@is.nagoya-u.ac.jp

**Abstract**

Unravelings are a class of transformations of conditional term rewriting systems into unconditional systems. Such transformations have been used to analyze and simulate conditional rewrite steps by unconditional rewrite steps for properties like (operational) termination. In this paper, we show how to prove confluence of conditional term rewriting systems via unravelings.

## 1 Introduction and Overview

Conditional term rewriting systems (CTRSs) are term rewriting systems in which rules may be constrained by equations over terms. Such systems arise naturally in many settings like functional programming and they have been used in applications like program inversion [8].

Yet, CTRSs are more difficult to analyze and many criteria that hold for unconditional TRSs do not hold for CTRSs. Therefore, several transformations have been defined that eliminate conditions in CTRSs [6, 14, 1, 12].

There are some results on confluence of CTRSs like [2, 13], yet there are no results known to us that use transformations to prove confluence of CTRSs.

The main difficulty in using transformations to prove confluence is that the transformed TRS may give rise to derivations that are not possible in the original CTRS. Another difficulty is that in order to encode conditions, the signature of the transformed system is different from the signature of the original CTRS. This might lead to derivations in which terms occur that are not defined in the original system.

In this paper, we show how to prove confluence of CTRSs via *unravelings*, the simplest class of transformations of CTRSs into TRSs. We focus on so-called *oriented, deterministic 3-CTRSs*, a class of CTRSs in which extra variables are allowed to a certain extent. We will use common notions and notations, like they are used in e.g. [10].

## 2 Unravelings

Unravelings are a class of transformations of CTRSs into unconditional TRSs that have been introduced in [6]. They have been the subject of interest in several publications [8, 4, 9, 5].

In an unraveling, a conditional rule is split into several unconditional rules. The conditions are encoded in new function symbols, called *U-symbols*, along with some variables. If the conditions are satisfied, then the rhs of the original conditional rule is reproduced.

*This work is supported by the Austrian Science Fund (FWF) international project I963 and the Japan Society for the Promotion of Science.

In the unraveling $\mathbb{U}_{seq}$, that is defined in [10] (based on [7]), one new function symbol is introduced for each condition in a conditional rule. By sequentially encoding the conditions, this unraveling can transform deterministic CTRSs (DCTRSs) into TRSs without extra variables. In DCTRSs, extra variables must occur on the rhs of a condition first so that their matchers can be determined by plain rewriting.

A conditional rule $\alpha : l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_k \to^* t_k$ is transformed into unconditional rules as follows:

$$\mathbb{U}_{seq}(\alpha) = \{\ \ l \to U_1^\alpha(s_1, \vec{X}_1),\ \ U_1^\alpha(t_1, \vec{X}_1) \to U_2^\alpha(s_2, \vec{X}_2),\ \ \ldots,\ \ U_k^\alpha(t_k, \vec{X}_k) \to r\ \ \}$$

where $X_i = \mathcal{V}ar(l, t_1, \ldots, t_{i-1})$. Here, $\vec{X}$ denotes the unique sequence of variables in $X$ under some fixed order on variables.

In order to distinguish terms in the unraveling that contain $U$-symbols, we will refer to such terms as *mixed terms* while we refer to terms without $U$-symbols as *original terms*.

It is easy to show that a derivation in a CTRS $u \to^*_{\mathcal{R}} v$ has a corresponding derivation in the unraveling $u \to^*_{\mathbb{U}_{seq}(\mathcal{R})} v$ (for all original terms $u, v$). This property is called *completeness*. While completeness is easy to prove and satisfied in general, its counterpart *soundness* only holds in certain cases (see e.g. [6, 4, 9, 5]).

# 3 Soundness for Joinability

We prove confluence of a CTRS $\mathcal{R}$ via derivations in the unraveling of $\mathcal{R}$ in the following way: for all original terms $s, t_1, t_2$ such that $t_1 \leftarrow^*_{\mathcal{R}} s \to^*_{\mathcal{R}} t_2$, we know by completeness that $t_1 \leftarrow^*_{\mathbb{U}(\mathcal{R})} s \to^*_{\mathbb{U}(\mathcal{R})} t_2$; if there is an original term $u$ such that $t_1 \to^*_{\mathbb{U}(\mathcal{R})} u \leftarrow^*_{\mathbb{U}(\mathcal{R})} t_2$, then by soundness we obtain that $t_1 \to^*_{\mathcal{R}} u \leftarrow^*_{\mathcal{R}} t_2$ in the original CTRS $\mathcal{R}$.

One difficulty in this approach is that we need to prove that $t_1$ and $t_2$ have a common descendant in $\mathbb{U}(\mathcal{R})$ that is an original term. We will therefore use another notion of soundness:

**Definition 1** (Soundness for joinability)**.** *An unraveling $\mathbb{U}$ is* sound for joinability *of a CTRS $\mathcal{R}$ if for all original terms $s, t$ such that $s \downarrow_{\mathbb{U}(\mathcal{R})} t$, also $s \downarrow_{\mathcal{R}} t$.*

We can use soundness for joinability to prove confluence for every CTRS:

**Lemma 2.** *Let $\mathcal{R}$ be a DCTRS and $\mathbb{U}$ be an unraveling. If $\mathbb{U}(\mathcal{R})$ is confluent and $\mathbb{U}$ is sound for joinability of $\mathcal{R}$, then $\mathcal{R}$ is confluent.*

*Proof.* Consider two terms $s, t$ such that $s \leftrightarrow^*_{\mathcal{R}} t$. Completeness of $\mathbb{U}$ implies $s \leftrightarrow^*_{\mathbb{U}(\mathcal{R})} t$. Since $\mathbb{U}(\mathcal{R})$ is confluent, therefore $s \downarrow_{\mathbb{U}(\mathcal{R})} t$, and by soundness of joinability $s \downarrow_{\mathcal{R}} t$. □

Although there is a strong connection between soundness and soundness for joinability, soundness does not imply soundness for joinability in general:

**Example 3.** Consider the following CTRS $\mathcal{R}$ that contains one conditional rule that is unraveled into two unconditional rules (using $\mathbb{U}_{seq}$):

$$\mathbb{U}_{seq}\left(\left\{\begin{array}{c} \mathsf{a} \to \mathsf{c} \to \mathsf{e} \\ \searrow\!\nearrow \quad \searrow \\ \mathsf{b} \to \mathsf{d} \to \mathsf{k} \\ \mathsf{f}(x) \to x \Leftarrow x \to^* \mathsf{e} \\ \mathsf{g}(x,x) \to \mathsf{h}(x,x) \\ \mathsf{h}(\mathsf{d},x) \to \mathsf{A}(x) \end{array}\right\}\right) = \left\{\begin{array}{ll} \vdots & \\ \mathsf{f}(x) \to U_1^\alpha(x,x), & U_1^\alpha(\mathsf{e},x) \to x \\ \vdots & \end{array}\right\}$$

$\mathbb{U}_{seq}$ is sound for non-erasing 2-DCTRSs ([5, Theorem 18] and [9, Corollary 5.5]), therefore the unraveling is sound. In $\mathbb{U}_{seq}(\mathcal{R})$, terms $\mathsf{g}(\mathsf{f}(\mathsf{a}),\mathsf{f}(\mathsf{b}))$ and $\mathsf{A}(\mathsf{f}(\mathsf{k}))$ are joinable:

$$\mathsf{g}(\mathsf{f}(\mathsf{a}),\mathsf{f}(\mathsf{b})) \to^* \mathsf{h}(U_1^\alpha(\mathsf{c},\mathsf{d}),U_1^\alpha(\mathsf{c},\mathsf{d})) \to^* \mathsf{h}(\mathsf{d},U_1^\alpha(\mathsf{c},\mathsf{d})) \to \mathsf{A}(U_1^\alpha(\mathsf{c},\mathsf{d})) \to^* \mathsf{A}(U_1^\alpha(\mathsf{k},\mathsf{k})) \leftarrow \mathsf{A}(\mathsf{f}(\mathsf{k}))$$

In $\mathcal{R}$, $\mathsf{A}(\mathsf{f}(\mathsf{k}))$ is irreducible, therefore $\mathsf{g}(\mathsf{f}(\mathsf{a}),\mathsf{f}(\mathsf{b})) \downarrow_{\mathcal{R}} \mathsf{A}(\mathsf{f}(\mathsf{k}))$ only if $\mathsf{g}(\mathsf{f}(\mathsf{a}),\mathsf{f}(\mathsf{b})) \to^*_{\mathcal{R}} \mathsf{A}(\mathsf{f}(\mathsf{k}))$. Yet, for this we need some common reduct $s$ of $\mathsf{f}(\mathsf{a})$ and $\mathsf{f}(\mathsf{b})$ such that $s \to^*_{\mathcal{R}} \mathsf{d}$ and $s \to^*_{\mathcal{R}} \mathsf{f}(\mathsf{k})$, but there is no original term satisfying these properties.

In the previous example, the derivation in $\mathbb{U}_{seq}(\mathcal{R})$ contains mixed terms. In order to prove soundness for joinability, we use a mapping $\mathsf{t}$ that translates terms in $\mathbb{U}(\mathcal{R})$ (including mixed terms) into original terms. Using such a translation we obtain a more general soundness criterion: an unraveling $\mathbb{U}$ is sound w.r.t. $\mathsf{t}$ for a DCTRS $\mathcal{R}$, if for all original terms $u$ and all mixed terms $v'$ such that $u \to^*_{\mathbb{U}(\mathcal{R})} v'$, $\mathsf{t}(v')$ is defined and $u \to^*_{\mathcal{R}} \mathsf{t}(v')$. Soundness w.r.t. $\mathsf{t}$ implies soundness for joinability:

**Lemma 4.** *If an unraveling $\mathbb{U}$ is sound w.r.t. $\mathsf{t}$ for a DCTRS $\mathcal{R}$, then $\mathbb{U}$ is also sound for joinability of $\mathcal{R}$.*

*Proof.* Let $s,t$ be two original terms such that there is some (possibly mixed) term $u'$ such that $s \to^*_{\mathbb{U}(\mathcal{R})} u' \leftarrow^*_{\mathbb{U}(\mathcal{R})} t$. Then, soundness w.r.t. $\mathsf{t}$ implies $s \to^*_{\mathcal{R}} \mathsf{t}(u') \leftarrow^*_{\mathcal{R}} t$. □

# 4 A New Unraveling

For many CTRSs, in particular overlay CTRSs, $\mathbb{U}_{seq}$ returns a non-confluent TRS so that we cannot use $\mathbb{U}_{seq}$ to prove confluence of the original CTRSs.

**Example 5** ([11])**.** The following CTRS defines *even* and *odd* predicates for natural number encoded by $\mathsf{0}$ and $\mathsf{s}$:

$$\mathcal{R}_{\mathsf{even}} = \left\{ \begin{array}{ll} \mathsf{even}(\mathsf{0}) \to \mathsf{true} & \mathsf{odd}(\mathsf{0}) \to \mathsf{false} \\ \mathsf{even}(\mathsf{s}(x)) \to \mathsf{false} \Leftarrow \mathsf{odd}(x) \to^* \mathsf{true} & \mathsf{odd}(\mathsf{s}(x)) \to \mathsf{false} \Leftarrow \mathsf{even}(x) \to^* \mathsf{true} \\ \mathsf{even}(\mathsf{s}(x)) \to \mathsf{true} \Leftarrow \mathsf{odd}(x) \to^* \mathsf{false} & \mathsf{odd}(\mathsf{s}(x)) \to \mathsf{true} \Leftarrow \mathsf{even}(x) \to^* \mathsf{false} \end{array} \right\}$$

The CTRS is unraveled into the following TRS using $\mathbb{U}_{seq}$:

$$\mathbb{U}_{seq}(\mathcal{R}_{\mathsf{even}}) = \left\{ \begin{array}{ll} \mathsf{even}(\mathsf{0}) \to \mathsf{true} & \mathsf{odd}(\mathsf{0}) \to \mathsf{false} \\ \mathsf{even}(\mathsf{s}(x)) \to U_1^\alpha(\mathsf{odd}(x),x) & \mathsf{odd}(\mathsf{s}(x)) \to U_1^\gamma(\mathsf{even}(x),x) \\ U_1^\alpha(\mathsf{true},x) \to \mathsf{false} & U_1^\gamma(\mathsf{true},x) \to \mathsf{false} \\ \mathsf{even}(\mathsf{s}(x)) \to U_1^\beta(\mathsf{odd}(x),x) & \mathsf{odd}(\mathsf{s}(x)) \to U_1^\eta(\mathsf{even}(x),x) \\ U_1^\beta(\mathsf{false},x) \to \mathsf{true} & U_1^\eta(\mathsf{false},x) \to \mathsf{true} \end{array} \right\}$$

The unraveled TRS is not confluent, for instance $\mathsf{even}(\mathsf{s}(\mathsf{0}))$ rewrites to $U_1^\alpha(\mathsf{odd}(\mathsf{s}(\mathsf{0}),\mathsf{0})$ and $U_1^\beta(\mathsf{odd}(\mathsf{s}(\mathsf{0}),\mathsf{0})$ that are not joinable. Note that a more complicated and practical example with the non-confluence problem can be found in [10, Example 7.2.49].

The following new unraveling returns a confluent TRS for certain overlay CTRSs. It strongly resembles the unraveling $\mathbb{U}_{seq}$, but we introduce new $U$-symbols based on the lhs of the transformed rule and terms in the conditions:

**Definition 6** (New unraveling)**.** *Let $\alpha$ be a conditional rule $l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_k \to^* t_k$ $(k \geq 0)$, then its unraveling is defined as*

$$\mathbb{U}_{conf}(\alpha) = \left\{ \begin{array}{r} l \to U_{l,s_1}(s_1, \vec{X}_1) \\ U_{l,s_1}(t_1, \vec{X}_1) \to U_{l,s_1,t_1,s_2}(s_2, \vec{X}_2) \\ \vdots \\ U_{l,s_1,t_1,\ldots,t_{k-1},s_k}(t_k, \vec{X}_k) \to r \end{array} \right\}$$

*where $X_i = \mathcal{V}ar(l, t_1, \ldots, t_{i-1})$. Note that for $U_\alpha$ and $U_{\alpha'}$, we use the same symbol, e.g., $U_\alpha$, if $\alpha'$ is a renamed variant of $U_\alpha$. The unraveled TRS $\mathbb{U}_{conf}(\mathcal{R})$ then is $\bigcup_{\alpha \in \mathcal{R}} \mathbb{U}_{conf}(\alpha)$.*

In order to prove soundness for joinability of certain cases, we use the following backtranslation that is also used in [5]:

$$\begin{array}{rll} \mathsf{tb}(x) = x & & \text{for all variables } x \\ \mathsf{tb}(U_{l,\ldots}(v, \vec{X}_i\sigma)) = l\,\mathsf{tb}(\sigma) & & \text{for all U-symbols } U_{l,\ldots} \\ \mathsf{tb}(f(t_1, \ldots, t_{ar(f)})) = f(\mathsf{tb}(t_1), \ldots, \mathsf{tb}(t_{ar(f)})) & & \text{for all non-U-symbols } f \end{array}$$

$\mathbb{U}_{seq}$ is sound w.r.t. $\mathsf{tb}$ for weakly left-linear CTRSs and since $\mathsf{tb}$ is well-defined for $\mathbb{U}_{conf}$ we can adapt the proof of [5, Theorem 3.28] to $\mathbb{U}_{conf}$.

**Lemma 7.** *$\mathbb{U}_{conf}$ is sound w.r.t. $\mathsf{tb}$ for a weakly left-linear CTRSs.*

*Proof (Sketch).* Since $\mathsf{tb}$ is well-defined and derivations in the conditions can be extracted from the $U$-symbols, we can use the proof of [5, Theorem 3.28]. □

**Corollary 8.** *$\mathbb{U}_{conf}$ is sound for joinability of weakly left-linear CTRSs.*

Finally, we obtain our main result:

**Theorem 9.** *A weakly left-linear DCTRS $\mathcal{R}$ is confluent if so is $\mathbb{U}_{conf}(\mathcal{R})$.*

**Example 10.** Consider the CTRS of Example 5. Its unraveling for $\mathbb{U}_{conf}$ has two rule less:

$$\mathbb{U}_{conf}(\mathcal{R}_{\mathsf{even}}) = \left\{ \begin{array}{r} \mathsf{even}(\mathsf{0}) \to \mathsf{true} \\ \mathsf{even}(\mathsf{s}(x)) \to U_{\mathsf{even}(\mathsf{s}(x)),\mathsf{odd}(x)}(\mathsf{odd}(x), x) \\ U_{\mathsf{even}(\mathsf{s}(x)),\mathsf{odd}(x)}(\mathsf{true}, x) \to \mathsf{false} \\ U_{\mathsf{even}(\mathsf{s}(x)),\mathsf{odd}(x)}(\mathsf{false}, x) \to \mathsf{true} \\ \mathsf{odd}(\mathsf{0}) \to \mathsf{false} \\ \mathsf{odd}(\mathsf{s}(x)) \to U_{\mathsf{odd}(\mathsf{s}(x)),\mathsf{even}(x)}(\mathsf{even}(x), x) \\ U_{\mathsf{odd}(\mathsf{s}(x)),\mathsf{even}(x)}(\mathsf{true}, x) \to \mathsf{false} \\ U_{\mathsf{odd}(\mathsf{s}(x)),\mathsf{even}(x)}(\mathsf{false}, x) \to \mathsf{true} \end{array} \right\}$$

The unraveled TRS is now confluent. It follows from left-linearity of $\mathbb{U}_{seq}(\mathcal{R}_{\mathsf{even}})$ that $\mathcal{R}_{\mathsf{even}}$ is weakly left-linear [5]. Therefore, by Theorem 9, $\mathcal{R}_{\mathsf{even}}$ is confluent.

To show the usefulness of our approach, we want to repeat a result of [13] using Theorem 9:

**Corollary 11.** *Orthogonal properly oriented right-stable 3-CTRSs are confluent.*

*Proof.* Orthogonal properly oriented right-stable 3-CTRSs are unraveled into orthogonal and therefore confluent TRSs by $\mathbb{U}_{conf}$. Therefore, we can apply Theorem 9. □

# 5 Conclusion and Perspectives

We have shown that unravelings can be used to prove confluence of CTRSs. In order to do this, we use soundness for joinability and a new unraveling, similar to the unraveling of [10], but with better properties concerning confluence while retaining soundness properties.

In the future, we want to show soundness for joinability of other classes of CTRSs and also use other transformations to analyze soundness properties.

A way to show joinability is the use of tree automata techniques developed to analyze reachability. The techniques are well investigated for TRSs, and they are very useful. However, the direct application of the techniques to CTRSs is very complicated and the constructed tree automata are often overapproximations (cf. [3]). Thus, unravelings would be very useful to analyze reachability and then confluence of CTRSs for which unravelings are sound. For this reason, we will also work for soundness of unravelings, e.g., to find soundness conditions.

# References

[1] S. Antoy, B. Brassel, and M. Hanus. Conditional narrowing without conditions. In *Proceedings of PPDP 2003*, pp. 20–31, ACM Press, 2003.

[2] J. Avenhaus and C. Loría-Sáenz. On conditional rewrite systems with extra variables and deterministic logic programs. In *Proceedings of LPAR 1994*, volume 822 of *Lecture Notes in Computer Science*, pp. 215–229, Springer, 1994.

[3] G. Feuillade and T. Genet. Reachability in conditional term rewriting systems. *Electronic Notes in Theoretical Computer Science*, 86(1):133–146, 2003.

[4] K. Gmeiner, B. Gramlich, and F. Schernhammer. On (un)soundness of unravelings. In *Proceedings of RTA 2010*, volume 6 of *Leibniz International Proceedings in Informatics*, pp. 119–134, 2010.

[5] K. Gmeiner, B. Gramlich, and F. Schernhammer. On soundness conditions for unraveling deterministic conditional rewrite systems. In *Proceedings of RTA 2012*, volume 15 of *Leibniz International Proceedings in Informatics*, pp. 193–208, 2012.

[6] M. Marchiori. Unravelings and ultra-properties. In *Proceedings of ALP 1996*, volume 1139 of *Lecture Notes in Computer Science*, pp. 107–121, Springer, 1996.

[7] M. Marchiori. On deterministic conditional rewriting. Technical Report MIT LCS CSG Memo no. 405, MIT, Cambridge, 1997.

[8] N. Nishida, M. Sakai, and T. Sakabe. Partial inversion of constructor term rewriting systems. In *Proceedings of RTA 2005*, volume 3467 of *Lecture Notes in Computer Science*, pp. 264–278, Springer, 2005.

[9] N. Nishida, M. Sakai, and T. Sakabe. Soundness of unravelings for deterministic conditional term rewriting systems via ultra-properties related to linearity. *Logical Methods in Computer Science*, 8(3):1–49, 2012.

[10] E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002.

[11] G. Roşu. From conditional to unconditional rewriting. In *Proceedings of WADT 2004*, volume 3423 of *Lecture Notes in Computer Science*, pp. 218–233, Springer, 2004.

[12] T.-F. Şerbănuţă and G. Roşu. Computationally equivalent elimination of conditions. In *Proceedings of RTA 2006*, volume 4098 of *Lecture Notes in Computer Science*, pp. 19–34, Springer, 2006.

[13] T. Suzuki, A. Middeldorp, and T. Ida. Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In *Proceedings of RTA 1995*, volume 914 of *Lecture Notes in Computer Science*, pp. 179–193, Springer, 1995.

[14] P. Viry. Elimination of conditions. *Journal of Symbolic Computation*, 28(3):381–401, 1999.

# A Confluent Pattern Calculus with Hedge Variables

Sandra Alves[1], Besik Dundua[1,3], Mário Florido[1] and Temur Kutsia[2]

[1] DCC-FC & LIACC, University of Porto, Portugal
[2] RISC, Johannes Kepler University, Linz, Austria
[3] VIAM, Ivane Javakhishvili Tbilisi State University, Georgia

**Abstract**

The dynamic pattern calculus described in this paper integrates the functional mechanism of the lambda-calculus and the capabilities of pattern matching with hedge variables, i.e., variables that can be instantiated by any finite sequence of terms. We propose a generic confluence proof, where the way pattern abstractions are applied in a non-deterministic calculus is axiomatized.

## 1 Introduction

There have been several approaches to design pattern calculi, extending the $\lambda$-calculus with pattern matching. One can mention the $\lambda\phi$-calculus [14], Pure Pattern Calculus [9], the $\rho$-calculus [5], and the typed versions thereof, e.g., [3].

Pattern calculi are expressive, however, there is a price to pay for that: confluence is lost and various restrictions have to be imposed to recover it. Cirstea and Faure in [4] proposed a generic confluence proof for dynamic pattern calculi with unitary matching algorithm. There are some conditions the matching function should satisfy, in order the guarantee the confluence. Pattern matching with hedge variables has interesting applications in programming languages [16], rewriting [8], knowledge representation [12], logic [11], etc.

The idea of transforming a non-deterministic calculus into a deterministic one with the help of sums is not new. It has been exploited in a way or another in, e.g., differential $\lambda$-calculus [7], linear-algebraic $\lambda$-calculus [1, 6], resource calculus [13]. The idea closest to us is described in [6], as a non-deterministic extension of the call-by-value $\lambda$-calculus, which corresponds to the additive fragment of the linear-algebraic $\lambda$-calculus. The + there is associative, commutative, and distributes over application both from right (corresponds to parallel composition) and left (corresponds to call-by-value). There is also the neutral element for the sum, expressing the impossible computation. In our calculus, sums originate from sets of different matchers. Hence, besides being associative and commutative, + is also idempotent. We do not have the neutral element: This would correspond to the case when there is no matcher between a pattern and a term. But in this case the reduction is not possible. We do not introduce a term to express this impossible reduction in the language itself. Like the sum in [6], our + is also left and right distributive over applications. Moreover, our language has the other features as well: Patterns, unranked symbols, and hedge variables, which really make a difference.

In this paper, we introduce yet another dynamic pattern calculus, which permits the use of hedge variables. These are variables which can be instantiated by finite, possibly empty, sequences of terms, called hedges.

# 2 The Calculus

In this section, we define the syntax and the operational semantics of the untyped dynamic pattern calculus with hedge variables and propose conditions under which the calculus is confluence.

## 2.1 Syntax

*Terms* are defined by the following grammar:

$$M, N ::= x \mid \ f \mid (M\,N) \mid (M\,X) \mid \lambda_{\mathcal{V}} M.N \mid M + N$$

where $(M\,N)$ is an *application of a term to a term* and $(M\,X)$ is an *application of a term to a hedge variable*. In $\lambda_{\mathcal{V}} M.N$ we call the term $M$ a *pattern*. We consider $+$ to be associative, commutative, and idempotent. Moreover, application distributes over $+$ both from the left and from the right. We write ACID for this property. We assume that terms are in the ACID normal form with respect to $+$ and application.

The letters $M, N, P, Q, W$ are used to denote terms, while $s$ is used for a hedge variable or a term. Application associates to the left, therefore we can write $(M\,s_1\,\cdots s_n)$ for $((\cdots(M\,s_1)\cdots)s_n)$. When there is no ambiguity, the outermost parentheses are omitted as well.

The *sets of free and bound variables* of a term $P$, denoted by $\mathtt{fv}(P)$ and by $\mathtt{bv}(P)$ respectively, are defined inductively as follows:

$$\begin{array}{ll}
\mathtt{fv}(x) = \{x\} \qquad \mathtt{fv}(f) = \varnothing & \mathtt{bv}(x) = \varnothing \qquad \mathtt{bv}(f) = \varnothing \\
\mathtt{fv}(M\,N) = \mathtt{fv}(M) \cup \mathtt{fv}(N) & \mathtt{bv}(M\,N) = \mathtt{bv}(M) \cup \mathtt{bv}(N) \\
\mathtt{fv}(M\,X) = \mathtt{fv}(M) \cup \{X\} & \mathtt{bv}(M\,X) = \mathtt{bv}(M) \\
\mathtt{fv}(\lambda_{\mathcal{V}} M.N)) = (\mathtt{fv}(M) \cup \mathtt{fv}(N)) \backslash \mathcal{V} & \mathtt{bv}(\lambda_{\mathcal{V}} M.N) = \mathtt{bv}(M) \cup \mathtt{bv}(N) \cup \mathcal{V} \\
\mathtt{fv}(M + N) = \mathtt{fv}(M) \cup \mathtt{fv}(N) & \mathtt{bv}(M + N) = \mathtt{bv}(M) \cup \mathtt{bv}(N)
\end{array}$$

The set $\mathcal{V}$ in the *abstraction* $\lambda_{\mathcal{V}} M.N$ is a subset of the set of free variables of $M$ and represents the set of variables bound by the abstraction. Note that, the set of variables bound by an abstraction is not necessarily the same as the set of free variables of the corresponding pattern. We adopt Barendregt's variable name convention [2] and identify terms modulo $\alpha$-equivalence.

*Hedges* are finite (possible empty) sequences of terms and hedge variables. We use $h$ to denote them. For the empty hedge we use $\epsilon$. For readability, we put hedges in angle brackets if they have more than one element, e.g., $\langle M, X, N\rangle$.

A *substitution* is a mapping from term variables to terms, and from hedge variables to hedges, such that all but finitely many term and hedge variables are mapped to themselves. We use $\varphi$ and $\vartheta$ to denote substitutions. Each substitution $\varphi$ is represented as a map $\{v_1 \mapsto \varphi(v_1), \ldots, v_n \mapsto \varphi(v_n)\}$ where the $v$'s are all those variables for which $\varphi(v_i) \neq v_i$. The sets $\mathit{Dom}(\varphi) = \{v_1, \ldots, v_n\}$ and $\mathit{Ran}(\varphi) = \{\varphi(v_1), \ldots, \varphi(v_n)\}$ are called the *domain* and the *range* of $\varphi$, respectively. The set $\mathit{Var}(\phi)$ is defined as $\mathit{Var}(\phi) = \mathit{Dom}(\varphi) \cup \mathtt{fv}(\mathit{Ran}(\varphi))$.

The *application* of a substitution $\varphi$ to a term $M$ replaces each *free* occurrence of a variable $v$ in $M$ with $\varphi(v)$. It is defined inductively:

$$\begin{array}{ll}
x\varphi = \varphi(x), \text{ if } x \in \mathit{Dom}(\varphi). & (M\,X)\varphi = M\varphi\,s_1, \ldots s_n, \\
x\varphi = x, \text{ if } x \notin \mathit{Dom}(\varphi). & \text{if } \varphi(X) = \langle s_1, \ldots, s_n\rangle.
\end{array}$$

$$f\varphi = f. \qquad (M\,X)\varphi = M\varphi \text{ if } \varphi(X) = \epsilon.$$
$$(M\,N)\varphi = M\varphi\,N\varphi. \qquad (M\,Y)\varphi = M\varphi\,Y \text{ if } Y \notin Dom(\varphi).$$
$$(M + N)\varphi = M\varphi + N\varphi. \qquad (\lambda_{\mathcal{V}} M.N)\varphi = \lambda_{\mathcal{V}} M\varphi.N\varphi.$$

In the abstraction, it is assumed that $Var(\varphi) \cap \mathcal{V} = \varnothing$.

The *composition* of substitutions, as well as the *restriction* of a substitution $\varphi$ to a set of variables $V$, denoted $\varphi|_V$, are defined in the standard way.

## 2.2 Reduction

Evaluation in the dynamic pattern calculus with hedge variables is given by three binary relations $\beta_{\mathsf{p}}$, $\mathsf{D_l}$, and $\mathsf{D_r}$ on terms, written in the form of reduction rules below. The relation $\beta_{\mathsf{p}}$ defines the way how pattern-abstractions are applied. It is parametrized by a function *Sol*, which takes as parameters two terms $M$ and $Q$ and a set of variables $\mathcal{V}$ and returns a finite set of substitutions. We denote it by $Sol(M \lessdot_{\mathcal{V}} Q)$. The relations $\mathsf{D_l}$ and $\mathsf{D_r}$ define how abstraction distributes over $+$:

$$\begin{aligned}
&\beta_{\mathsf{p}} : (\lambda_{\mathcal{V}} M.N)\,Q \to N\varphi_1 + \cdots + N\varphi_n, \text{where } M \text{ and } Q \text{ are not of the form}\\
&\qquad\qquad W_1 + W_2 \text{ and } Sol(M \lessdot_{\mathcal{V}} Q) = \{\varphi_1, \ldots, \varphi_n\},\ n \geqslant 1.\\
&\mathsf{D_l} : \lambda_{\mathcal{V}} M_1 + M_2.N \to \lambda_{\mathcal{V}} M_1.N + \lambda_{\mathcal{V}} M_2.N.\\
&\mathsf{D_r} : \lambda_{\mathcal{V}} M.N_1 + N_2 \to \lambda_{\mathcal{V}} M.N_1 + \lambda_{\mathcal{V}} M.N_2.
\end{aligned}$$

In what follows, $\to_P$ denotes the compatible closure of the union of $\beta_{\mathsf{p}}, \mathsf{D_l}$ and $\mathsf{D_r}$ relations and $\twoheadrightarrow_P$ denotes the reflexive and transitive closure of $\to_P$.

## 2.3 Confluence

The solution of a matching equation $P \lessdot_{\mathcal{V}} M$ is a substitution $\varphi$ such that $P\varphi = M$. We can reformulate matching procedures with hedge variables proposed in [10] to define the function *Sol*. Therefore, *Sol* accepts a set of equations of the form $P \lessdot_{\mathcal{V}} M$ and returns complete the set of solutions. The following example shows that the function *Sol* can lead to diverging reductions:

**Example 1.** *The term* $M = (\lambda_{\{Z\}} fZ.(\lambda_{\{X,Y\}} fXY.fX)fZ)fab$ *reduces to two different normal forms:*

1. $M \to_P (\lambda_Z fZ.(f + fZ))fab \to_P f + fab.$
2. $M \to_P (\lambda_{\{X,Y\}} fXY.fX)fab \to_P f + fa + fab.$

Our goal is to impose restrictions on *Sol* so that confluence is guaranteed. The confluence proof will be based on the standard method due to Tait and Martin-Löf [2]. It requires the notion of *parallel reduction* which is defined as follows:

$$\frac{}{s \Rightarrow_P s} \qquad \frac{s_1 \Rightarrow_P s'_1 \quad \ldots \quad s_n \Rightarrow_P s'_n}{\langle s_1, \ldots, s_n\rangle \Rightarrow_P \langle s'_1, \ldots, s'_n\rangle} \qquad \frac{M \Rightarrow_P M' \quad s \Rightarrow_P s'}{M\,s \Rightarrow_P M'\,s'}$$

$$\frac{M \Rightarrow_P M' \quad N \Rightarrow_P N'}{\lambda_{\mathcal{V}} M.N \Rightarrow_P \lambda_{\mathcal{V}} M'.N'} \qquad \frac{M_1 \Rightarrow_P M'_1 \quad M_2 \Rightarrow_P M'_2 \quad N \Rightarrow_P N'}{\lambda_{\mathcal{V}}(M_1 + M_2).N \Rightarrow_P \lambda_{\mathcal{V}} M'_1.N' + \lambda_{\mathcal{V}} M'_2.N'}$$

$$\frac{M \Rightarrow_P M' \quad N \Rightarrow_P N'}{M + N \Rightarrow_P M' + N'} \qquad \frac{M \Rightarrow_P M' \quad N_1 \Rightarrow_P N_1' \quad N_2 \Rightarrow_P N_2'}{\lambda_{\mathcal{V}} M.(N_1 + N_2) \Rightarrow_P \lambda_{\mathcal{V}} M'.N_1' + \lambda_{\mathcal{V}} M'.N_2'}$$

$$\frac{M \Rightarrow_P M' \quad N \Rightarrow_P N' \quad Q \Rightarrow_P Q'}{(\lambda_{\mathcal{V}} M.N)Q \Rightarrow_P N'\varphi_1 + \cdots + N'\varphi_n}, \text{ where } Sol(M' \lessdot_{\mathcal{V}} Q') = \{\varphi_i \mid 1 \leqslant i \leqslant n\}$$

Note that the parallel reduction is compatible. Its definition is extended to substitutions having the same domain by setting $\varphi \Rightarrow_P \varphi'$ if for all $v \in Dom(\varphi) = Dom(\varphi')$, we have $v\varphi \Rightarrow_P v\varphi'$.

Now we impose conditions on *Sol* so that confluence is guaranteed.

**$\mathbf{H}_0$:** If $\varphi \in Sol(P \lessdot_{\mathcal{V}} M)$, then $Dom(\varphi) = \mathcal{V}$ and $\mathtt{fv}(Ran(\varphi)) \subseteq \mathtt{fv}(M)$.

**$\mathbf{H}_1$:** If $Sol(P \lessdot_{\mathcal{V}} M) = \{\varphi_1, \ldots, \varphi_n\}$, $n \geqslant 1$, then for all $\vartheta$ with $Var(\vartheta) \cap \mathcal{V} = \varnothing$, we have $Sol(P\vartheta \lessdot_{\mathcal{V}} M\vartheta) = \{(\varphi_1\vartheta)|_{\mathcal{V}}, \ldots, (\varphi_n\vartheta)|_{\mathcal{V}}\}$.

**$\mathbf{H}_2$:** If $Sol(P \lessdot_{\mathcal{V}} M) = \{\varphi_1, \ldots, \varphi_n\}$, $n \geqslant 1$, $P \Rightarrow_P P'$, and $M \Rightarrow_P M'$, then $Sol(P' \lessdot_{\mathcal{V}} M') = \{\varphi_1', \ldots, \varphi_m'\}$, $m \geqslant 1$, and

(a) for all $1 \leqslant i \leqslant n$ there exists $1 \leqslant j \leqslant m$ such that $\varphi_i \Rightarrow_P \varphi_j'$ and

(b) for all $1 \leqslant j \leqslant m$ there exists $1 \leqslant i \leqslant n$ such that $\varphi_i \Rightarrow_P \varphi_j'$.

These conditions extend the ones for the core dynamic pattern calculus from [4]. We will show that they are sufficient for proving confluence of our calculus. We assume that the relations $\rightarrow_P$ and $\Rightarrow_P$ in the lemmas and in the theorem below use a *Sol* which satisfies $\mathbf{H}_0$, $\mathbf{H}_1$, and $\mathbf{H}_2$. We do not state this explicitly in the conditions.

Looking back to Example 1, is it not surprising that confluence does not hold there: *Sol* used in that example violates the $\mathbf{H}_1$ property. Just take $P = f(X, Y), M = f(Z), \mathcal{V} = \{X, Y\}$, and $\vartheta = \{Z \mapsto \langle a, b\rangle\}$. Then we get $Sol(P \lessdot_{\mathcal{V}} M) = \{\{X \mapsto \epsilon, Y \mapsto Z\}, \{X \mapsto Z, Y \mapsto \epsilon\}\}$ and $Sol(P\vartheta \lessdot_{\mathcal{V}} M\vartheta) = \{\{X \mapsto \epsilon, Y \mapsto \langle a, b\rangle\}, \{X \mapsto a, Y \mapsto b\}, \{X \mapsto \langle a, b\rangle, Y \mapsto \epsilon\}\}$.

**Lemma 1.** *The following inclusion hold:* $\rightarrow_P \subseteq \Rightarrow_P \subseteq \twoheadrightarrow_P$.

**Lemma 2** (Fundamental Lemma)**.** *For all terms $M$ and $M'$ and for all substitutions $\varphi$ and $\varphi'$ with $Dom(\varphi) = Dom(\varphi')$, if $M \Rightarrow_P M'$ and $\varphi \Rightarrow_P \varphi'$, then $M\varphi \Rightarrow_P M'\varphi'$.*

**Lemma 3** (Diamond Property of Parallel Reduction)**.** *For all terms $M$, $N$, and $Q$, if $M \Rightarrow_P N$ and $M \Rightarrow_P Q$, then there exists a term $W$ such that $N \Rightarrow_P W$ and $Q \Rightarrow_P W$.*

**Theorem 1.** *For all terms $M, N$, and $Q$, if $M \twoheadrightarrow_P N$ and $M \twoheadrightarrow_P Q$ then there exists a term $W$ such that $N \twoheadrightarrow_P W$ and $Q \twoheadrightarrow_P W$.*

This theorem shows that under the condition of *Sol* satisfying $\mathbf{H}_0$, $\mathbf{H}_1$, and $\mathbf{H}_2$, the relation $\twoheadrightarrow_P$ (and, hence, the calculus) is confluent.

## Acknowledgements

This research has been partially supported by the FCT fellowship (ref. SFRH/BD/62058/2009), by the Austrian Science Fund (FWF) under the project SToUT (P 24087-N18), and the Rustaveli Science Foundation under the grant FR/611/4-102/12.

# References


[1] P. Arrighi and G. Dowek. Linear-algebraic lambda-calculus: higher-order, encodings, and confluence. In Voronkov [15], pages 17–31.

[2] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 1984. Revised edition.

[3] G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. Pure patterns type systems. In A. Aiken and G. Morrisett, editors, *POPL*, pages 250–261. ACM, 2003.

[4] H. Cirstea and G. Faure. Confluence of pattern-based calculi. In F. Baader, editor, *RTA*, volume 4533 of *LNCS*, pages 78–92. Springer, 2007.

[5] H. Cirstea and C. Kirchner. The rewriting calculus - parts I and II. *Logic Journal of the IGPL*, 9(3), 2001.

[6] A. Díaz-Caro and B. Petit. Linearity in the non-deterministic call-by-value setting. In C.-H. L. Ong and R. J. G. B. de Queiroz, editors, *WoLLIC*, volume 7456 of *LNCS*, pages 216–231. Springer, 2012.

[7] T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 309(1-3):1–41, 2003.

[8] F. Jacquemard and M. Rusinowitch. Closure of hedge-automata languages by hedge rewriting. In Voronkov [15], pages 157–171.

[9] C. B. Jay and D. Kesner. Pure pattern calculus. In P. Sestoft, editor, *ESOP*, volume 3924 of *LNCS*, pages 100–114. Springer, 2006.

[10] T. Kutsia. Solving equations with sequence variables and sequence functions. *J. Symb. Comput.*, 42(3):352–388, 2007.

[11] T. Kutsia and B. Buchberger. Predicate logic with sequence variables and sequence function symbols. In A. Asperti, G. Bancerek, and A. Trybulec, editors, *MKM*, volume 3119 of *LNCS*, pages 205–219. Springer, 2004.

[12] C. Menzel. Knowledge representation, the world wide web, and the evolution of logic. *Synthese*, 182(2):269–295, 2011.

[13] M. Pagani and S. R. D. Rocca. Solvability in resource lambda-calculus. In C.-H. L. Ong, editor, *FOSSACS*, volume 6014 of *LNCS*, pages 358–373. Springer, 2010.

[14] V. van Oostrom. Lambda calculus with patterns. Technical Report IR-228, Vrije Universiteit, Amsterdam, 1990.

[15] A. Voronkov, editor. *Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings*, volume 5117 of *LNCS*. Springer, 2008.

[16] S. Wolfram. *The Mathematica Book*. Wolfram Media, 5th edition, 2003.

# Synchronizing Applications of the Parallel Moves Lemma to Formalize Confluence of Orthogonal TRSs in PVS*

Ana Cristina Rocha Oliveira[1], André Luiz Galdino[2] and Mauricio Ayala-Rincón[1]

[1] Grupo de Teoria da Computação, Departamentos de Ciência da Computação e Matemática
Universidade de Brasília, Brasília D.F., Brazil
[2] Departamento de Matemática
Universidade Federal de Goiás - Campus Catalão, Goiânia, Brazil
anacrismarie@gmail.com, galdino@unb.br, ayala@unb.br

**Abstract**

A complete formalization in PVS of the theorem of confluence of orthogonal term rewriting systems is presented. The formalized proof uses the PVS *theory* `trs` maintaining its distinguishing feature of remaining close to textbook's proofs. The proof is based on a formalization of the Parallel Moves Lemma. Auxiliary lemmas are given which use an inductive construction of the crucial positions of a term originating a parallel divergence. Classifying all these positions, by application of the parallel moves lemma to the crucial divergence subterms, the desired common term of joinability is built.

## 1 Introduction

The formalization of confluence of orthogonal TRSs consists of a *theory* called `orthogonality` that imports the PVS *theory* `trs` [GAR09], that is available in the NASA LaRC PVS library [trs13], and which includes formalizations of several rewriting results ranging from specification of basic rewriting notions and properties to more elaborated results such as the Critical Pair theorem [GAR10], confluence and modular properties of abstract reduction systems [GAR08] and completeness of first-order unification algorithms [AGdMAR11]. The *theory* `orthogonality` specifies notions such as `Ambiguous?(E)`, `Left_Linear?(E)`, `Orthogonal?(E)` and `parallel_reduction?(E)`, all them of interest for dealing with formalization of theorems about orthogonal TRS's (see [ROAR13]).

Proofs of confluence of orthogonal TRSs have been known at least since Rosen's seminal work [Ros73] which is based on the well-known Parallel Moves Lemma (for short, PML). This proof was adapted by Huet in [Hue80], for proving the confluence of left-linear and parallel closed TRSs that admit critical pairs joinable from left to right in a sole step of parallel reduction. To the best of our knowledge, the sole related formalization was developed very recently in Isabelle/HOL in [Thi12], where unlike orthogonality, weak orthogonality, that allows the existence of trivial critical pairs, is assumed. The chapter on orthogonality of [BKdV03] surveys different styles of proofs of confluence of orthogonal TRSs, that are not different in essence. Our style of proof follows the inductive approach in [BN98] which depends on the analysis of properties of the parallel rewriting relation and the PML. In this analysis, the PML is applied for guaranteeing parallel joinability of each *principal* redex involved in a parallel divergence from a term $s$: $u \leftleftarrows s \rightrightarrows v$. These redices appear at positions $\pi$ in which on the one side

*Work supported by FAPDF PRONEX 2009/00091-0 grant. First, second and third authors partially supported by a CAPES Ph.D. scholarship, an FAPEG research support grant and a CNPq research fellowship.

one step of rewriting and, on the other side one step of parallel reduction are applied: either $u|_\pi \leftarrow t_\pi \rightrightarrows v|_\pi$ or $u|_\pi \leftleftarrows t_\pi \rightarrow v|_\pi$, being $t|_\pi$ an instance of the left-hand side (**lhs**, for short) of a rule (see Fig. 1). From our point of view this choice is very adequate, because the discipline of formalization that guided the development of the *theory* `trs`, that is also the one desired for `orthogonality`, is providing proofs that are as close as possible to textbook's proofs.

Regarding a previous work in which the general lines of the formalization of this theorem were presented [ROAR13], here we specifically report on the necessary analysis to *synchronize* applications of the PML to terms in order to join the divergence terms in a parallel divergence. This involves the construction of a sequence of dominating positions in which either the same rule was applied or on the one side a rewriting rule was applied and on the other side a parallel reduction was applied. Each term of the parallel divergence is obtained by simple rewriting steps at sequences of disjunct positions, using associated sequences of substitutions and rules. The theory is available at `www.mat.unb.br/~ayala/publications.html`.

## 2 Basic Notions and Definitions

Familiarity with rewriting notations and notions is assumed (c.f. [BN98, BKdV03]). Terms, built from a given signature and a set of enumerable variables, are represented as trees and positions of a term $t$ as sequences of naturals. For a position $\pi$ of a term $t$, $t|_\pi$ denotes the subterm at position $\pi$ and parallel positions $\pi$ and $\pi'$ are sequences such that neither $\pi$ is a prefix of $\pi'$ nor $\pi'$ is a prefix of $\pi$. Given a TRS $R$, the rewriting relation is denoted as $\rightarrow_R$, and $R$ is omitted when it is clear from the context. Composition of relations is denoted as $\circ$. The inverse of $\rightarrow$ is denoted by $\leftarrow$ and syntactic equality by $=$. The reflexive closure of the relation $\rightarrow$ is denoted as $\rightarrow^=$ and the reflexive transitive closure as $\rightarrow^*$. Similarly, ${}^*\leftarrow$ will denote the reflexive transitive closure of $\leftarrow$. The relations of local divergence, divergence and joinability are given by $\leftarrow \circ \rightarrow$, ${}^*\leftarrow \circ \rightarrow^*$ and $\rightarrow^* \circ {}^*\leftarrow$, respectively. One says that $\rightarrow$ is *confluent* if $({}^*\leftarrow \circ \rightarrow^*) \subseteq (\rightarrow^* \circ {}^*\leftarrow)$; and that it has the *diamond property* if $(\leftarrow \circ \rightarrow) \subseteq (\rightarrow \circ \leftarrow)$.

A rule $e = (l, r)$ is a pair of terms such that the lhs cannot be a variable and the variables occurring in its right-hand side (**rhs**, for short) should appear in the lhs. A TRS is given as a set of rules. The reduction relation $\rightarrow_E$ induced by a TRS $E$ is built as follows: a term $s$ reduces to $t$, denoted as $s \rightarrow t$, if $s = s[\pi \leftarrow lhs(e)\sigma] \rightarrow_E s[\pi \leftarrow rhs(e)\sigma] = t$, where, in general, $u[\pi \leftarrow v]$ denotes the term obtained from $u$ by replacing the subterm at position $\pi$ of $u$ by the term $v$.

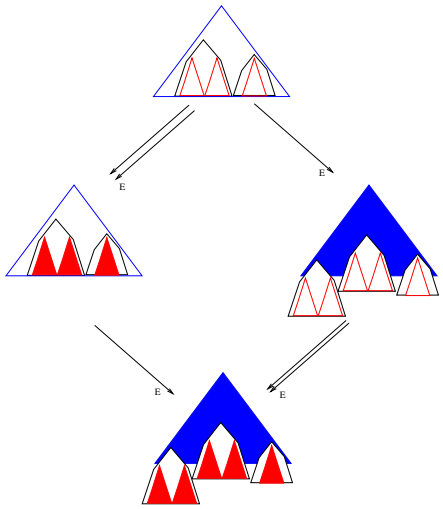



Figure 1: Parallel Moves Lemma (PML)

Another crucial relation is *parallel reduction*: one says that $s$ reduces in parallel to $t$, denoted as $s \rightrightarrows t$, if there exist finite sequences of the same length $n \geq 0$, $\Pi := \pi_1, \ldots, \pi_n$; $\Sigma := \sigma_1, \ldots, \sigma_n$ and $\Gamma := e_1, \ldots, e_n$ of parallel positions of $s$, substitutions and rules, respectively, such that: $\forall 1 \leq i \leq n : s|_{\pi_i} = lhs(e_i)\sigma_i$, and $t$ is obtained from $s$, by replacing all subterms

at positions in $\Pi$ by $t|_{\pi_i} = rhs(e_i)\sigma_i$. All this is summarized by the following notation: $s = s[\pi_1 \leftarrow l_1\sigma_1, \ldots, \pi_n \leftarrow l_n\sigma_n] \rightrightarrows_E s[\pi_1 \leftarrow r_1\sigma_1, \ldots, \pi_n \leftarrow r_n\sigma_n] = t$, where, $l_i = lhs(e_i)$ and $r_i = rhs(e_i)$, for $1 \leq i \leq n$. For short, notation $s \rightrightarrows s[\Pi \leftarrow rhs(\Gamma)\Sigma]$ is used and one will say that $s$ reduces in parallel through reductions at positions $\Pi$ using $\Sigma$-instances of rules $\Gamma$.

By simple analysis one has that $\rightarrow \subseteq \rightrightarrows \subseteq \rightarrow^*$. Thus, $\rightarrow^* = \rightrightarrows^*$, from which proving the diamond property for $\rightrightarrows$ will provide a proof of confluence of $\rightarrow$. Thus, the crucial result to be formalized is the theorem below.

**Theorem 2.1** (Orthogonality implies diamond property)**.** *Let $R$ be a TRS orthogonal. Then, the relation $\rightrightarrows$ has the diamond property.*

Orthogonal TRSs have no critical pairs and are left-linear. The PML (see Fig. 1) states that for all instances of rules of an orthogonal system, say $(l, r)\sigma$, if $l\sigma \rightrightarrows s$, then there exists a $t$ such that $r\sigma \rightrightarrows t \leftarrow s$.

Essentially, the proof is based on the observation that since instances of lhs's of other rules can only overlap at variable positions of $l$, then by left-linearity one has that $s = l\sigma'$ for some substitution $\sigma'$. Also, for all variables $x\sigma \rightrightarrows x\sigma'$. Thus, one has $l\sigma \rightrightarrows l\sigma'$ and $r\sigma \rightrightarrows r\sigma' \leftarrow l\sigma$.

In order to prove the diamond property of orthogonal TRSs (see Fig. 2), the structure of a parallel divergence should be stratified in such a way that crucial positions are chosen to apply either the PML or non ambiguity (inexistence of CPs). A parallel divergence from a term $s$ through positions $\Pi_1$ and $\Pi_2$ correspondingly using $\Sigma_1$ and $\Sigma_2$-instances of rules $\Gamma_1$ and $\Gamma_2$, has the form $t_1 = s[\Pi_1 \leftarrow rhs(\Gamma_1)\Sigma_1] \leftrightarrows s \rightrightarrows s[\Pi_2 \leftarrow rhs(\Gamma_2)\Sigma_2] = t_2$, where $\Pi_i, \Gamma_i, \Sigma_i$, for $i = 1, 2$, are sequences of parallel positions of $s$, rules and substitutions, respectivelly, as in the definition of parallel reduction.

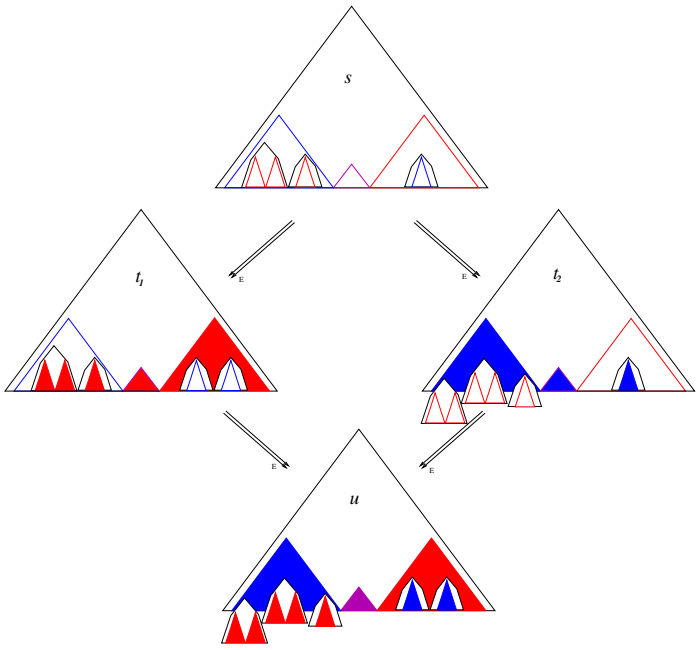



Figure 2: Diamond property of parallel rewriting

To prove theorem 2.1, a term $u$ should be built such that $t_1 \rightrightarrows u \leftrightarrows t_2$.

# 3 Formalization

The formalization of the Diamond Property of parallel reduction of orthogonal TRSs is done by induction on the length of crucial positions occurring in a parallel divergence. These positions are built through the specification of an inductive operator `Pos_Over`$(\Pi_1, \Pi_2)$ that builds the

subsequence of positions from $\Pi_1$ that are parallel to all positions in $\Pi_2$ or that have positions in the sequence $\Pi_2$ below them. Thus, the crucial positions of a divergence are given by a sequence $\Pi$ carefully constructed and corresponding to the sequence $\texttt{Pos\_Over}(\Pi_1, \Pi_2) \cup \texttt{Pos\_Over}(\Pi_2, \Pi_1) \cup (\Pi_1 \cap \Pi_2)$, where by $\cup$ and $\cap$ of sequences one means the sequence obtained by concatenation and by including only common members, respectively. Intuitively, it is easy to check (see Fig. 2) that exactly the subterms at these positions are the ones modified in the parallel divergence and that joining the subterms at these positions of $t_1$ and $t_2$ will provide the joinability term $u$. But synchronizing the one-step-parallel movements from the subterms at these positions of $t_1$ and $t_2$, that is essentially building the necessary premisses to apply the PML, requires a great deal of technical work which corresponds to a significant part of the whole formalization.

To guarantee that these crucial positions $\Pi$, as mentioned above, are the ones modified when a term $s$ parallelly diverges to $t_1$ and $t_2$, the lemma `replace_par_pos_dominance` was proved. This lemma shows that if $s \rightrightarrows t_i$ through reductions at positions $\Pi_i$, for $i = 1, 2$ correspondingly, then one is able to write $t_i$ as $s[\Pi \leftarrow (t_i|_\pi)_{\pi \in \Pi}]$. This is possible because for every position $\pi'$ in $\Pi_i$, there exists a position $\pi$ in $\Pi$ that is (above or) prefix of $\pi'$, what is satisfied by $\Pi$ indeed.

Regarding joinability of subterms at positions $\pi$ in $\Pi$ of $t_1$ and $t_2$, two cases are to be analyzed. Firstly, the subterms of $t_1$ and $t_2$ at a position $\pi$ in $\Pi_1 \cap \Pi_2$ are easily joined in one step of parallel reduction because these subterms are identical since there are no critical pairs in an orthogonal TRS. Secondly, the lemma `divergence_in_Pos_Over` shows explicitly how the divergence in the subterms of $s, t_1$ and $t_2$ at a position $\pi$ in $\texttt{Pos\_Over}(\Pi_1, \Pi_2) \cup \texttt{Pos\_Over}(\Pi_2, \Pi_1)$ satisfies the conditions required by the PML. So subterms $t_1|_\pi$ and $t_2|_\pi$ are joinable in one step of parallel reduction too (see Fig 1).

So, by the discussion in the last paragraph, for all $\pi$ in the sequence of crucial positions $\Pi$, there exists $u_\pi$ such that $t_1|_\pi \rightrightarrows u_\pi \lleftarrow t_2|_\pi$ whenever $t_1 \lleftarrow s \rightrightarrows t_2$ through reductions at positions $\Pi_1$ and $\Pi_2$. A sequence of terms $U := (u_\pi)_{\pi \in \Pi}$ is built such that replacing subterms of $s$ at these positions gives the required term of parallel joinability: $u = s[\Pi \leftarrow U]$. At this point it is necessary to stress that a great deal of effort was necessary to formalize the synchronization the positions, rules and substitutions of $\Pi_i, \Gamma_i$ and $\Sigma_i$, for $i = 1, 2$, involved in the construction of the terms $u_\pi$ and in general in the construction of the joinability term $u$.

Another important lemma to conclude the proof of diamond property of parallel reduction is `parallel_reduction_context`, which was necessary because one cannot guarantee directly that, if $t_1|_\pi \rightrightarrows u_\pi$, $\forall \pi \in \Pi$, then $t_1 \rightrightarrows u$. The formalization of this lemma uses induction on the length of the sequence of crucial positions $\Pi$ and, through it, it is possible to conclude that $t_1 = s[\Pi \leftarrow (t_1|_\pi)_{\pi \in \Pi}] \rightrightarrows s[\Pi \leftarrow U] = u \lleftarrow s[\Pi \leftarrow (t_2|_\pi)_{\pi \in \Pi}] = t_2$.

Excluding all these technical details necessary to adequately apply the PML, the formalization of confluence of orthogonal TRSs follows the sketch of proof presented in Section 6.4 of [BN98]. In its current status, the PVS *theory* `orthogonality` has about 53000 lines of proofs and 770 lines of specification. It is worth mentioning that the proof file includes also typing information that substantially increases over the part strictly related with the formalization.

# 4 Related work and Conclusions

The PVS *theory* `orthogonality` includes a complete formalization of the theorem of confluence of orthogonal TRSs which is based on the PML. Although the formalization follows the lines of textbook's proofs such as the one given in [BN98], its development required a great deal of invisible effort that was necessary to adequately apply the PML. Several additional lemmas were formalized in order to prove that a parallel divergence can be structured as an ordered

sequence of crucial divergences from subterms of the term of divergence, from which instances of the hypotheses of the PML are detected. Then, applying the PML to these subterms one builds a general term of joinability in parallel. The formalization of the PML was finished obtaining in this way a complete formalization of the theorem of confluence of orthogonal TRSs.

Future developments and extensions are related with the usability of this formalization to check automatically confluence of functional specifications that follow the discipline of orthogonality. Also, an interesting investigation is related with formalization of confluence of several variants such as weak orthogonal TRSs and Church-Rosser theorems for variants of the $\lambda$-calculus. Adaptation of the proof style used here to the alternative definition of parallel reduction used in the short proof of confluence for variants of the $\lambda$-calculus in [Tak95], and extensible for orthogonal TRSs as it is done in the chapter on orthogonality in [BKdV03], is also of great interest. Other technologies of proof surveyed in [BKdV03] as the one based on developments [vO97] as well as strengthening the current result to the permutation equivalence [HL91], that were pertinently pointed out by the reviewers, deserve formalizations, but although parts of the current formalization can be reused, they will require formalization developments substantially different from the current one.

# References


[AGdMAR11] A.B. Avelar, A.L. Galdino, F.L.C. de Moura, and M. Ayala-Rincón. A Formalization of the Theorem of Existence of First-Order Most General Unifiers. In *Proceedings 6th Workshop on Logical and Semantic Frameworks with Applications, LSFA'11*, volume 81 of *EPTCS*, pages 63–78, 2011.

[BKdV03] M. Bezem, J.W. Klop, and R. de Vrijer, editors. *Term Rewriting Systems by TeReSe*. Number 55 in Cambridge Tracts in Theoretical Computer Science. Cambridge UP, 2003.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and* All That. Cambridge UP, 1998.

[GAR08] A. L. Galdino and M. Ayala-Rincón. A Formalization of Newman's and Yokouchi Lemmas in a Higher-Order Language. *Journal of Formalized Reasoning*, 1(1):39–50, 2008.

[GAR09] A. L. Galdino and M. Ayala-Rincón. A PVS *Theory* for Term Rewriting Systems. In *Proceedings of the Third Workshop on Logical and Semantic Frameworks, with Applications - LSFA'08*, volume 247 of *ENTCS*, pages 67–83, 2009.

[GAR10] A. L. Galdino and M. Ayala-Rincón. A Formalization of the Knuth-Bendix(-Huet) Critical Pair Theorem. *J. of Automated Reasoning*, 45(3):301–325, 2010.

[HL91] G. P. Huet Huet and J.-J. Lévy. Computations in Orthogonal Rewriting Systems, I. In *Computational Logic - Essays in Honor of Alan Robinson*, pages 395–414, 1991.

[Hue80] G. Huet. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *J. of the ACM*, 27(4):797–821, October 1980.

[ROAR13] A. C. Rocha Oliveira and M. Ayala-Rincón. Formalizing the confluence of orthogonal rewriting systems. *CoRR*, abs/1303.7335, 2013.

[Ros73] B. K. Rosen. Tree-manipulating systems and church-rosser theorems. *J. of the ACM*, 20(1):160–187, 1973.

[Tak95] M. Takahashi. Parallel Reductions in lambda-Calculus. *Inf. Comput.*, 118(1):120–127, 1995.

[Thi12] R. Thiemann. Certification of Confluence Proofs using CeTA. In *First International Workshop on Confluence (IWC 2012)*, page 45, 2012.

[trs13] Theory `trs`. Available at `http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library`, NASA LaRC PVS library, consulted March 2013.

[vO97] V. van Oostrom. Developing Developments. *Theor. Comput. Sci.*, 175(1):159–181, 1997.

# KBCV 2.0 – Automatic Completion Experiments*

Thomas Sternagel

University of Innsbruck, Innsbruck, Austria
thomas.sternagel@uibk.ac.at

**Abstract**

This paper describes the automatic mode of the new version of the Knuth-Bendix Completion Visualizer. The internally used data structures have been overhauled and the performance was dramatically improved by introducing caching, parallelization, and term-indexing in the computation of critical pairs and simplification. The new version is much faster and can complete three more systems.

## 1 Introduction

The *Knuth-Bendix Completion Visualizer* (KBCV) is an interactive/automatic tool for Knuth-Bendix completion and equational logic proofs. The basic functions of the previous release are described in detail in [5, 7]. This paper addresses implementation issues to improve the performance of the automatic completion mode and reports on experiments of the new release KBCV 2.0. The tool is available under the *GNU Lesser General Public License 3* at

`http://cl-informatik.uibk.ac.at/software/kbcv`

In the sequel we assume familiarity with term rewriting, and completion [1]. Nevertheless we recall the basics.

Completion is a procedure which takes as input a (finite) set of equations $\mathcal{E}$ and a reduction order $>$ (or it tries to construct this reduction order on the fly with the help of an external termination tool, see [8]) and attempts to construct a terminating and confluent term rewrite system (TRS) $\mathcal{R}$ with the same equational theory as $\mathcal{E}$. In case the completion procedure succeeds, two terms are equivalent with respect to $\mathcal{E}$ if and only if they reduce to the same normal form with respect to $\mathcal{R}$, that is, $\mathcal{R}$ represents a decision procedure for the word problem of $\mathcal{E}$.

The computation is done by generating a finite sequence of intermediate TRSs which constitute approximations of the equational theory of $\mathcal{E}$. Following Bachmair and Dershowitz [2] the completion procedure can be modeled as an inference system (see Figure 1). The inference rules work on pairs $(\mathcal{E}, \mathcal{R})$ where $\mathcal{E}$ is a finite set of equations and $\mathcal{R}$ is a finite set of rewrite rules. The goal is to transform an initial pair $(\mathcal{E}, \varnothing)$ into a pair $(\varnothing, \mathcal{R})$ such that $\mathcal{R}$ is terminating, confluent and equivalent to $\mathcal{E}$. In our setting a completion procedure based on these rules may succeed (find $\mathcal{R}$ after finitely many steps), loop, or fail. In Figure 1 a reduction order $>$ is provided as part of the input. We use $s \xrightarrow{\exists}_{\mathcal{R}} u$ to express that $s$ is reduced by a rule $\ell \to r \in \mathcal{R}$ such that $\ell$ cannot be reduced by another rule with left-hand side $s$. The notation $s \mathrel{\dot{\approx}} t$ denotes either of $s \approx t$ and $t \approx s$.

KBCV internally uses indexed equations $i\colon l \approx r$ and rules $j\colon l \to r$, where $i$ and $j$ are unique positive integers and $l$ and $r$ are terms, called the left- and right-hand side respectively.

*Supported by the Austrian Science Fund (FWF) international project I963 and the Japan Society for the Promotion of Science.

$$\text{DEDUCE}\ \frac{(\mathcal{E},\mathcal{R})}{(\mathcal{E}\cup\{s\approx t\},\mathcal{R})}\ \text{if}\ s\ {}_{\mathcal{R}}\!\leftarrow u\rightarrow_{\mathcal{R}} t \qquad \text{ORIENT}\ \frac{(\mathcal{E}\cup\{s\mathrel{\dot{\approx}} t\},\mathcal{R})}{(\mathcal{E},\mathcal{R}\cup\{s\to t\})}\ \text{if}\ s>t$$

$$\text{COMPOSE}\ \frac{(\mathcal{E},\mathcal{R}\cup\{s\to t\})}{(\mathcal{E},\mathcal{R}\cup\{s\to u\})}\ \text{if}\ t\rightarrow_{\mathcal{R}} u \qquad \text{DELETE}\ \frac{(\mathcal{E}\cup\{s\approx s\},\mathcal{R})}{(\mathcal{E},\mathcal{R})}$$

$$\text{COLLAPSE}\ \frac{(\mathcal{E},\mathcal{R}\cup\{s\to t\})}{(\mathcal{E}\cup\{u\approx t\},\mathcal{R})}\ \text{if}\ s\overset{\sqsupset}{\rightarrow}_{\mathcal{R}} u \qquad \text{SIMPLIFY}\ \frac{(\mathcal{E}\cup\{s\mathrel{\dot{\approx}} t\},\mathcal{R})}{(\mathcal{E}\cup\{u\mathrel{\dot{\approx}} t\},\mathcal{R})}\ \text{if}\ s\rightarrow_{\mathcal{R}} u$$

Figure 1: The inference rules of *completion*.

## 2 Optimizing Automatic Completion

KBCV 2.0 is implemented in Scala 2.10.0,[1] an object-functional programming language which compiles to Java bytecode. For this reason KBCV is portable and runs on Windows and Linux machines. The developed term library (`scala-termlib`, available from KBCV's homepage) was completely overhauled and consists of approximately 2100 lines of code. The new KBCV builds upon this library and has an additional 5000 lines of code.

The main goal for this release was to improve the performance of KBCV especially in automatic mode. Some more details on the automatic mode can be found in [5, Section 5.2.1] and [7, Section 2.2]. Looking at the flow chart of the automatic mode depicted in Figure 2 we first had to identify critical parts, where speed-up would be possible.

1. The procedure starts in the SIMPLIFY-phase, where both sides of equations are rewritten as far as possible.
2. Trivial equations, that is, equations where both sides are the same are dropped in the DELETE-phase.
3. The third phase checks if $\mathcal{E}$ is empty and if all critical pairs between left-hand sides of rules in $\mathcal{R}$ are joinable.[2]
4. Then the procedure chooses a single equation which it tries to orient. The used heuristic is to select an equation where the length of the left- and right-hand sides is minimal. The cost for orientation mainly depends on the used termination tool.
5. Now in the COMPOSE-phase the procedure simplifies all right-hand sides of rules as far as possible.
6. After that, in the COLLAPSE-phase, it tries to simplify left-hand sides of rules.
7. Finally DEDUCE computes critical pairs and adds them to the set of equations.

From this assessment we see that (2) is trivial and already very fast and (4) mainly depends on an external program. So we focus on the remaining phases. In the sequel we will sometimes refer to (3) and (7) collectively as *critical pair computation* and to (1), (5), and (6) as *simplification*.

[1] `http://www.scala-lang.org/`

[2] The computation and check for joinability of critical pairs is only needed because of KBCV's interactive mode in which inference rules may be fired in an arbitrary order.

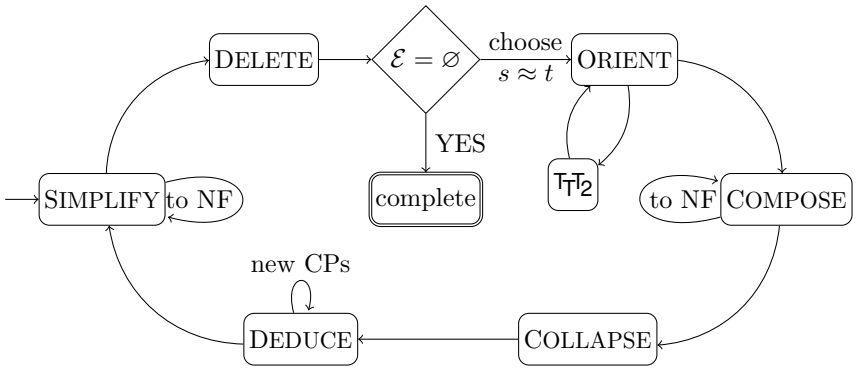



Figure 2: KBCV's automatic completion procedure.

The first idea (which was already partly implemented in versions 1.7 and 1.8 of KBCV) was to prevent re-computation by introducing caching. Next the independent parts of the procedure were parallelized to make the most of modern multi-core/processor architectures. Finally we also introduced term-indexing in order to speed up unification and matching of terms. These steps are described in some more detail in the next three sections. We compare the resulting speed-ups for various combinations of these methods in Section 3.

## 2.1 Caching

In order to avoid redundancy in critical pair computations and simplifications we introduced four new data structures for caching.

Each time a critical pair is computed KBCV stores the pair of indices of the overlapping rules which caused the new equation. The next time automatic completion has to compute critical pairs (in phases (3) or (7) of the procedure) it only computes critical pairs from overlaps which are not already stored.

We use three different caches for COMPOSE, COLLAPSE, and SIMPLIFY respectively. The first cache has an entry for each rule. In this entry we store the set of indices of rules which have already been tried to simplify this rule. Next time automatic completion has to simplify a rule it only tries the rules which are not cached yet. The other two caches work just in the same way.

## 2.2 Parallelization

While automatic completion (Figure 2) executes the single phases sequentially, within a phase there are completely independent computations which can be parallelized.

- DEDUCE: The computation of critical pairs.
- COMPOSE: The composition of rules.
- COLLAPSE: The collapsing of rules.
- SIMPLIFY: The simplification of equations.

In order to get the most out of modern multi-core architectures we re-implemented those four phases. Now each single step (e.g. the computation of critical pairs between two particular

| | KBCV-b-i-u | KBCV-i-u | KBCV-b-u | KBCV-b-i | KBCV-u | KBCV-i | KBCV-b | KBCV |
|---|---|---|---|---|---|---|---|---|
| *completed* | 85 | 87 | 85 | 85 | 89 | 90 | 85 | 90 |
| *total time* | 1142.6 | 512.8 | 498.0 | 384.5 | 1163.4 | 1321.3 | 321.8 | 1116.2 |
| *avg. time* | 13.4 | 5.9 | 5.9 | 4.5 | 13.1 | 14.7 | 3.8 | 12.4 |
| AD93_Z22 | | 83.9 | | | 44.7 | 41.8 | | 32.9 |
| BGK94_D16 | | 30.5 | | | 25.7 | 25.6 | | 23.2 |
| BGK94_Z22W | | | | | 598.7 | 220.6 | | 201.6 |
| LS94_G1 | | | | | | 583.6 | | 514.5 |
| SK90_3.09 | | | | | 168.1 | 161.1 | | 90.1 |

Table 1: Experimental results on 115 systems, timeout: 600s.

rules, or one specific rewrite step on one side of an equation) are separate computations which can be handled by a pool of worker threads. The main program waits until all results are computed and then continues with the non-parallel part of the procedure.

### 2.3 Term Indexing

Both unification of terms (needed for the computation of critical pairs) and matching (needed for rewriting of terms) can get very expensive for large systems with large left-hand sides of rules. To counteract that we now store the left-hand sides of rules in a discrimination tree (see for example [4]) which allows for very fast filtering of so called *candidate sets* (which are typically very small). Getting a unifiable or matching term from this candidate set is much faster than checking all left-hand sides of rules.

## 3 Experiments

The experiments we describe here were carried out on a 64bit GNU/Linux machine with 48 AMD Opteron™ 6174 processors and 315 GB of RAM. The kernel version is 2.6.32. The version of Java on this machine is 1.7.0_03. For the JVM we limited the stack size for each thread to 10MB, set the initial heap size to 1GB, and the maximum heap size to 2GB. The test-bed we worked with consists of 115 systems from the distribution of MKBTT.[3] KBCV was launched using the following flags:

```
./kbcv -a -p -s 600 -m "./ttt2 -cpf xml - 1" <inputfile>
```

Here the `-a` flag tells KBCV to switch to automatic mode, `-p` causes KBCV to output the CPF proof of completion on `stdout`, `-s 600` sets the timeout to 600 seconds and finally `-m` sets the termination-check method to use, in our case calls to the external termination tool TTT2. There are three more flags we used in the experiments: `-b` disables caching, `-i` disables term-indexing, and `-u` disables parallelization. The tool instances where parallelization was enabled used all of the 48 processors.

The upper part of Table 1 gives the number of completed systems, the total time needed to complete them and the average time for each of the completed systems for different configurations of KBCV. The lower part lists systems which only certain configurations of KBCV could complete together with the time. The detailed experiments are available online.[4] Here each

[3] http://cl-informatik.uibk.ac.at/software/mkbtt/index.php

[4] http://cl-informatik.uibk.ac.at/software/kbcv/experiments/kbcv2/

column is labeled with KBCV plus the set flags. So the first column labeled KBCV-b-i-u gives the results for KBCV without caching, term-indexing, and parallelization, while the last column shows the results for KBCV using all three methods. What we see is that without optimization KBCV can complete 85 out of the 115 systems and the average time for that is 13.4 seconds per system. Without caching (columns three, four, and seven) we are not able to complete additional systems, although we achieve a speed-up of about 2.6 using only term-indexing or parallelization, and 3.5 using both of these methods. Only using caching (column two) already establishes two more systems with a speed-up for the initial 85 systems of about 2.9. Caching plus term-indexing (column five) already yields two more successful systems and a speed-up with respect to the 85 systems of about 3.5. If we combine caching with parallelization (column six) we get yet another system and a speed-up for the initial systems of about 4.0. Finally KBCV using all three methods achieves a speed-up of 4.5 for the initial 85 systems. All found proofs have been certified by CeTA [6].

# 4 Conclusion

Three different methods to enhance KBCV's automatic completion procedure have been investigated and compared. We have seen that these methods, most notably caching, achieve a huge performance boost for the automatic completion procedure of KBCV 2.0.

If we look at the 115 systems we tested, we see that most of them only consist of about 10 to 20 rules and that the left-hand sides of those are also pretty small. When we work with much larger systems with more complicated left-hand sides parallelization and term-indexing become more and more important. We for example tried to only compute critical pairs for a subset of the HOL Light [3] simpset (about 3000) rules. Without parallelization we had to cancel the experiment after several days. Using parallelization KBCV was able to compute the 300,000 critical pairs in less than two hours.

A next step to further push the procedure would be to investigate different heuristics for the selection of equations in the ORIENT-phase.

# References

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

[2] Leo Bachmair and Nachum Dershowitz. Equational inference, canonical proofs, and proof orderings. *J. ACM*, 41:236–276, 1994.

[3] John Harrison. HOL Light: A tutorial introduction. In *FMCAD*, pages 265–269, 1996.

[4] I. V. Ramakrishnan, R. C. Sekar, and Andrei Voronkov. Term Indexing. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1853–1964. Elsevier and MIT Press, 2001.

[5] Thomas Sternagel. Automatic proofs in equational logic. Master's thesis, University of Innsbruck, 2012.

[6] Thomas Sternagel, René Thiemann, Harald Zankl, and Christian Sternagel. Recording completion for finding and certifying proofs in equational logic. In *IWC'12*, pages 31–36, 2012.

[7] Thomas Sternagel and Harald Zankl. KBCV - Knuth-Bendix completion visualizer. In *Proceedings of the 6th International Joint Conference on Automated Reasoning*, volume 7364 of *LNAI*, pages 530–536. Springer-Verlag, 2012.

[8] Ian Wehrman, Aaron Stump, and Edwin Westbrook. Slothrop: Knuth-Bendix completion with a modern termination checker. In *RTA'06*, pages 287–296. Springer-Verlag, 2006.

# Confluent Let-Floating

Clemens Grabmayer[1] and Jan Rochel[2]

[1] Department of Philosophy, Utrecht University
clemens@phil.uu.nl
[2] Department of Computing Sciences, Utrecht University
jan@rochel.info

**Abstract**

We develop a rewrite analysis for floating (moving) let-bindings in expressions of $\boldsymbol{\lambda}_{\mathsf{letrec}}$, the $\lambda$-calculus with the construct letrec that is denoted by let (as in the programming language Haskell). In particular we consider a HRS (higher-order rewrite system) for let-lifting, which moves let-bindings upward, and another HRS for let-sinking, which moves let-bindings downward. We show confluence and termination of the let-lifting and let-sinking rewrite systems, yielding the existence of unique normal forms. Our confluence proofs use a critical pair analysis and the critical pair theorem to establish local confluence, and the termination of these systems to obtain confluence by applying Newman's Lemma.

Let-floating is an operation employed by transformations that simplify and optimize program code as part of compilers of functional languages. For example the lambda-lifting transformation of functional programs into supercombinators contains a step called 'let-floating' [4, 15.5.4] or 'block-floating' [1], in which let-bindings are floated out (upward, we call it 'let-lifting'). Lambda-lifting transforms a let-block-structured program into a set of recursive equations whose right-hand sides are supercombinators. This transformation has an inverse called lambda-dropping [1], which contains the step 'block-sinking' in which let-bindings are floated in (downward, we call it 'let-sinking'). The use of let-floating operations in either direction for optimizing and fine-tuning the execution behavior of compiled functional programs has been studied in [8].

As a more general concept, let-floating acts on expressions of $\boldsymbol{\lambda}_{\mathsf{letrec}}$, the $\lambda$-calculus with the construct letrec for formulating recursion and explicit substitution. We denote letrec as let like in the programming language Haskell (no confusion should arise with the non-recursive explicit-substitution construct let), but keep the symbol $\boldsymbol{\lambda}_{\mathsf{letrec}}$. In our terminology, 'floating' stands for movements in either direction, whereas 'lifting' and 'sinking' indicate upward and downward shifts in the syntax tree, respectively. Let-floating manipulates the structure of let-bindings in $\boldsymbol{\lambda}_{\mathsf{letrec}}$-expressions, but preserves the unfolding semantics of the expressions (the denoted infinite $\lambda$-terms). A let-binding-group $B$ can be lifted up toward the innermost $\lambda$-abstraction that has a free variable occurrence in $B$. A group of $n$ interdependent let-bindings $\vec{f} = \vec{F}(\vec{f})$ with $\vec{f} = \langle f_1, \ldots, f_n \rangle$ can be sunk until an applicative term is encountered where both in its function subterm and in its argument subterm some recursion variable $f_i$ with $i \in \{1, \ldots, n\}$ occurs.

Our interest in let-floating stems from an investigation of the relationship between $\boldsymbol{\lambda}_{\mathsf{letrec}}$-expressions and term graph representations for cyclic $\lambda$-terms [3]. Translations of $\boldsymbol{\lambda}_{\mathsf{letrec}}$-expressions into representing term graphs typically ignore the precise positioning of the let-bindings, and instead extract the cyclic structure of the term. Therefore such translations map $\boldsymbol{\lambda}_{\mathsf{letrec}}$-expressions that are related by let-floating to the same term graph. For the definition of (left-)inverses of such translations, it is desirable to obtain natural representatives of let-floating equivalence classes by restricting the direction of let-floating operations to upward or downward.

We develop a rewrite analysis of let-floating. When decomposed into locally applicable rewrite steps on $\boldsymbol{\lambda}_{\mathsf{letrec}}$-expression, let-floating operations typically move let-bindings upward or downward over applications and abstractions, or merge different let-binding groups, given that such steps do not interfere with the structure of the $\lambda$-bindings. We formalize $\boldsymbol{\lambda}_{\mathsf{letrec}}$-expressions

as higher-order rewriting system (HRS) terms [10], and define two HRSs that describe different kinds of let-floating transformations as rewrite systems: let-lifting for moving let-bindings upward, and let-sinking for moving them downward. In both cases let-bindings are split whenever necessary for moves, and merged whenever possible. We show confluence and termination of the let-lifting and let-sinking rewrite systems, and by that, unique normalization.

# 1 Let-lifting

We formulate expressions in (untyped) $\boldsymbol{\lambda}_{\mathsf{letrec}}$ as HRS-terms [10] over the signature $\{\mathsf{abs}, \mathsf{app}\} \cup \{\mathsf{let}_n\_\mathsf{in} \mid n \in \mathbb{N}\}$, where $\mathsf{abs} : (\mathsf{trm} \to \mathsf{trm}) \to \mathsf{trm}$, $\mathsf{app} : \mathsf{trm} \to \mathsf{trm} \to \mathsf{trm}$, and for all $n \in \mathbb{N}$, $\mathsf{let}_n\_\mathsf{in} : (\mathsf{trm}^n \to \mathsf{trm}^{n+1}) \to \mathsf{trm}$ over the base type $\mathsf{trm}$. As an example, consider the $\boldsymbol{\lambda}_{\mathsf{letrec}}$-term:

$$\lambda x.\, \mathbf{let}\ f = g,\ g = x\ \mathbf{in}\ f\,x \qquad\qquad \mathsf{abs}(x.\,\mathsf{let}_2\_\mathsf{in}\,(fg.\,(g, x, \mathsf{app}(f, x))))$$

in familiar (first-order) notation and in a formulation as HRS-term. Here the index 2 in the symbol $\mathsf{let}_2\_\mathsf{in}$ indicates the number of bindings in the binding group of the let-expression. While building on this HRS-formulation, we will generally use the familiar syntax for let-expressions.

We consider five schemes of rules for lifting let-bindings, see below. A step according to a rule from $({}_{\mathrm{let}}\!\nearrow @_0)$ or $({}_{\mathrm{let}}\!\nearrow @_1)$ lifts a let-binding-group over an application. In steps according to rules from $({}_{\mathrm{let}}\!\nearrow \lambda)$, a let-binding-group immediately below an abstraction is either lifted over the abstraction in its entirety, or it is split into a part that is lifted and a part that stays behind. Steps according to rules in $(\mathbf{let\text{-}in}_{\_\ \mathrm{let}}\!\nearrow)$ merge the binding-groups of two let-expressions where one forms the in-part of the other. A step according to rules from $(\mathbf{let}_{\_\ \mathrm{let}}\!\nearrow)$ lifts, out of its position, the binding-group $B'$ of a let-expression that defines a recursive variable $g$ in a let-binding-group $B$, merges $B$ with $B'$, and adapts the definition of $g$ accordingly. Sequences of steps due to (exchange)-rules can rearrange the order in which let-bindings occur in a binding-group.

$$(_{\mathrm{let}}\!\nearrow @_0)\quad (\mathbf{let}\ \vec{f} = \vec{F}(\vec{f})\ \mathbf{in}\ E_0(\vec{f}))\,E_1 \ \to\ \mathbf{let}\ \vec{f} = \vec{F}(\vec{f})\ \mathbf{in}\ E_0(\vec{f})\,E_1$$

$$(_{\mathrm{let}}\!\nearrow @_1)\quad E_0\,(\mathbf{let}\ \vec{f} = \vec{F}(\vec{f})\ \mathbf{in}\ E_1(\vec{f})) \ \to\ \mathbf{let}\ \vec{f} = \vec{F}(\vec{f})\ \mathbf{in}\ E_0\,E_1(\vec{f})$$

$$(_{\mathrm{let}}\!\nearrow \lambda)\quad \lambda x.\,\mathbf{let}\ \vec{f} = \vec{F}(\vec{f}),\ \vec{g} = \vec{G}(\vec{f}, \vec{g}, x)\ \mathbf{in}\ E(\vec{f}, \vec{g}, x)$$
$$\to \begin{cases} \mathbf{let}\ \vec{f} = \vec{F}(\vec{f})\ \mathbf{in}\ \lambda x.\,E(\vec{f}, x) & \text{if } \vec{g} \text{ is empty} \\ \mathbf{let}\ \vec{f} = \vec{F}(\vec{f})\ \mathbf{in}\ \lambda x.\,\mathbf{let}\ \vec{g} = \vec{G}(\vec{f}, \vec{g}, x)\ \mathbf{in}\ E(\vec{f}, \vec{g}, x) & \text{if neither } \vec{f} \text{ nor } \vec{g} \text{ are empty} \end{cases}$$

$$(\mathbf{let\text{-}in}_{\_\ \mathrm{let}}\!\nearrow)\quad \mathbf{let}\ \vec{f} = \vec{F}(\vec{f})\ \mathbf{in}\ \mathbf{let}\ \vec{g} = \vec{G}(\vec{f}, \vec{g})\ \mathbf{in}\ E(\vec{f}, \vec{g}) \to\ \mathbf{let}\ \vec{f} = \vec{F}(\vec{f}),\ \vec{g} = \vec{G}(\vec{f}, \vec{g})\ \mathbf{in}\ E(\vec{f}, \vec{g})$$

$$(\mathbf{let}_{\_\ \mathrm{let}}\!\nearrow)\quad \mathbf{let}\ \vec{f} = \vec{F}(\vec{f}, g),\ g = \mathbf{let}\ \vec{h} = \vec{H}(\vec{f}, g, \vec{h})\ \mathbf{in}\ G(\vec{f}, g, \vec{h})\ \mathbf{in}\ E(\vec{f}, g) \to\ \mathbf{let}\ \vec{f} = \vec{F}(\vec{f}, g),\ g = G(\vec{f}, g, \vec{h}),\ \vec{h} = \vec{H}(\vec{f}, g, \vec{h})\ \mathbf{in}\ E(\vec{f}, g)$$

$$(\text{exchange})\quad \mathbf{let}\ B_0,\ f_i = F_i(\vec{f}),\ f_{i+1} = F_{i+1}(\vec{f}),\ B_1\ \mathbf{in}\ E(\vec{f}) \to\ \mathbf{let}\ B_0,\ f_{i+1} = F_{i+1}(\vec{f}),\ f_i = F_i(\vec{f}),\ B_1\ \mathbf{in}\ E(\vec{f})$$

Here we have used the familiar syntax of let-expressions instead of the underlying HRS-syntax.[1]

[1] E.g. $\mathsf{app}((\mathsf{let}_n\_\mathsf{in}\,(\vec{y}.\,(x_1(\vec{y}), \ldots, x_n(\vec{y}), z_0(\vec{y})))), z_1) \ \to\ \mathsf{let}_n\_\mathsf{in}\,(\vec{y}.\,(x_1(\vec{y}), \ldots, x_n(\vec{y}), \mathsf{app}(z_0(\vec{y}), z_1)))$ are the rules of scheme $({}_{\mathrm{let}}\!\nearrow @_0)$ in HRS-notation with the leading abstractions $x_1 \ldots x_n z_0 z_1.$ on either side kept implicit.

Note that an alternative formulation of $({}_{\mathrm{let}}\nearrow\lambda)$ that only can lift a let-binding-group over an abstraction in its entirety, but that does not allow to split it, has a drawback. In order to obtain the same let-lifting rewrite relation, also a rule for splitting binding-groups is required, for example the converse of $(\mathbf{let\text{-}in}_{-\ \mathrm{let}}\nearrow)$. But then together with the rule $(\mathbf{let\text{-}in}_{-\ \mathrm{let}}\nearrow)$ itself, which is needed for confluence, avoidable non-termination is introduced in the let-lifting system (which is of a different kind than the non-termination caused by (exchange)-steps alone).

By $\boldsymbol{R}_{\mathrm{let}\nearrow}$ we denote the HRS consisting of the first five rules above. By $\boldsymbol{R}_{\mathrm{let}\nearrow\mathrm{ex}}$ we denote the HRS consisting of all six rules above, thus the extension of $\boldsymbol{R}_{\mathrm{let}\nearrow}$ with the rule (exchange). The rewrite relations of $\boldsymbol{R}_{\mathrm{let}\nearrow}$ and $\boldsymbol{R}_{\mathrm{let}\nearrow\mathrm{ex}}$ are denoted by ${}_{\mathrm{let}}\nearrow$ and ${}_{\mathrm{let}}\nearrow_{\mathrm{ex}}$, respectively. The rewrite relation $\to_{\mathrm{ex}}$ is induced by steps according to the rule (exchange), and $=_{\mathrm{ex}}$ is the convertibility relation with respect to $\to_{\mathrm{ex}}$. The *let-lifting rewrite relation* ${}_{\mathrm{let}}\nearrow$ *on* $\boldsymbol{\lambda}_{\mathsf{letrec}}$*-terms* is defined as the rewrite relation ${}_{\mathrm{let}}\nearrow$ modulo $=_{\mathrm{ex}}$, that is (see below), by ${}_{\mathrm{let}}\nearrow := =_{\mathrm{ex}}\cdot{}_{\mathrm{let}}\nearrow\cdot=_{\mathrm{ex}}$. For example:

$$\lambda x.(\mathbf{let}\ f=\mathbf{let}\ g=x\ \mathbf{in}\ g\ \mathbf{in}\ f)\,x\ {}_{\mathrm{let}}\nearrow\ \lambda x.(\mathbf{let}\ f=g,\ g=x\ \mathbf{in}\ f)\,x\ {}_{\mathrm{let}}\nearrow\ \lambda x.\mathbf{let}\ f=g,\ g=x\ \mathbf{in}\ f\,x$$

is a ${}_{\mathrm{let}}\nearrow$-rewrite sequence (and even a ${}_{\mathrm{let}}\nearrow$-rewrite sequence) to a normal form. Another final ${}_{\mathrm{let}}\nearrow$-step here yields the $=_{\mathrm{ex}}$-equivalent term $\lambda x.\mathbf{let}\ g=x,\ f=g\ \mathbf{in}\ f\,x$. Therefore ${}_{\mathrm{let}}\nearrow$ is not confluent. However, it will turn out that ${}_{\mathrm{let}}\nearrow$ is 'confluent modulo' $=_{\mathrm{ex}}$.

An *abstract equational rewrite system* $\mathcal{A}=\langle A,\to,\sim\rangle$ is an abstract rewrite system $\langle A,\to\rangle$ that is endowed with an equivalence relation $\sim$ on $A$. The rewrite relation $\to_{/\sim/}$ of $\to$ *modulo* $\sim$ is defined as $\to_{/\sim/} := \sim\cdot\to\cdot\sim$. The *class rewrite relation* $\to_{[\sim]}$ *of* $\to$ *with respect to* $\sim$ is induced by $\to_{/\sim/}$ on the $\sim$-equivalence classes on $A$ by: for all $a,b\in A$, $[a]_\sim\to_{[\sim]}[b]_\sim$ if and only if $a\to_{/\sim/}b$.

The rewrite relation $\to$ is called *locally confluent modulo* $\sim$ (resp. *confluent modulo* $\sim$) if it holds: $\leftarrow\cdot\to\subseteq\twoheadrightarrow\cdot\sim\cdot\twoheadleftarrow$ (resp. $\twoheadleftarrow\cdot\twoheadrightarrow\subseteq\twoheadrightarrow\cdot\sim\cdot\twoheadleftarrow$). The lemma below reduces confluence properties for $\to_{/\sim/}$ and $\to_{[\sim]}$ to corresponding properties of a rewrite relation subsumed by $\to_{/\sim/}$.

**Lemma 1.** *Let* $\langle A,\to,\sim\rangle$ *be an abstract equational rewrite system with* $\sim\ =\ \leftrightarrow_\sim^*$ *for a rewrite relation* $\to_\sim$ *on* $A$*. Then it holds: if* $\sim\cdot\to\ \cup\ \to_\sim$ *is locally confluent (confluent), then* $\to_{/\sim/}$ *is locally confluent modulo* $\sim$ *(confluent modulo* $\sim$*), and* $\to_{[\sim]}$ *is locally confluent (confluent).*

The *let-lifting rewrite relation* ${}_{[\mathrm{let}]}\nearrow$ *on* $=_{\mathrm{ex}}$*-equivalence classes of* $\boldsymbol{\lambda}_{\mathsf{letrec}}$*-terms* is defined as the class rewrite relation ${}_{[\mathrm{let}]}\nearrow := {}_{\mathrm{let}}\nearrow_{[=_{\mathrm{ex}}]}$ (note that ${}_{\mathrm{let}}\nearrow = {}_{\mathrm{let}}\nearrow_{/=_{\mathrm{ex}}/}$):

$$[L]_{=_{\mathrm{ex}}}\ {}_{[\mathrm{let}]}\nearrow\ [L']_{=_{\mathrm{ex}}}\ :\Longleftrightarrow\ L\ {}_{\mathrm{let}}\nearrow\ L'\qquad(\text{for all } \boldsymbol{\lambda}_{\mathsf{letrec}}\text{-terms } L,\ L')\ .$$

**Lemma 2.** ${}_{\mathrm{let}}\nearrow$ *is locally confluent modulo* $=_{\mathrm{ex}}$*, and* ${}_{[\mathrm{let}]}\nearrow$ *is locally confluent.*

*Proof (Outline).* We define a HRS $\boldsymbol{R}_{\mathrm{let}\nearrow\mathrm{ex}}$ with $=_{\mathrm{ex}}\cdot{}_{\mathrm{let}}\nearrow\ \cup\ \to_{\mathrm{ex}}$ as its rewrite relation, by extending $\boldsymbol{R}_{\mathrm{let}\nearrow_{\mathrm{ex}}}$ through adding, for each rule $\rho$ in $\boldsymbol{R}_{\mathrm{let}\nearrow}$, all variant rules $\rho_\phi$ with respect to $=_{\mathrm{ex}}$-permutation steps $=_{\mathrm{ex}}^{\phi}$ on the left-hand sides of the pattern of $\rho$. In this way each rule scheme $(\sigma)$ of $\boldsymbol{R}_{\mathrm{let}\nearrow}$ gives rise to a rule scheme $(\sigma)_{=_{\mathrm{ex}}}$ of $\boldsymbol{R}_{\mathrm{let}\nearrow\mathrm{ex}}$. Then every step $=_{\mathrm{ex}}^{\phi}\cdot\to_\rho$ for the rewrite relation $=_{\mathrm{ex}}\cdot{}_{\mathrm{let}}\nearrow$, where $\to_\rho$ is a step according to a rule $\rho$ of scheme $(\sigma)$ in $\boldsymbol{R}_{\mathrm{let}\nearrow}$, is a step $\to_{\rho_\phi}$ according to a variant rule $\rho_\phi$ of scheme $(\sigma)_{=_{\mathrm{ex}}}$ in $\boldsymbol{R}_{\mathrm{let}\nearrow\mathrm{ex}}$.

Now it can be checked that all critical pairs of $\boldsymbol{R}_{\mathrm{let}\nearrow\mathrm{ex}}$ are joinable. For example, solving a critical overlap between rules $({}_{\mathrm{let}}\nearrow@_0)$ in $({}_{\mathrm{let}}\nearrow@_0)_{=_{\mathrm{ex}}}$ and $({}_{\mathrm{let}}\nearrow@_1)$ in $({}_{\mathrm{let}}\nearrow@_1)_{=_{\mathrm{ex}}}$:

$$\begin{array}{ccc}
(\mathbf{let}\ \vec{f}=F(\vec{f})\ \mathbf{in}\ E_0(\vec{f}))\,(\mathbf{let}\ \vec{g}=G(\vec{g})\ \mathbf{in}\ E_1(\vec{g})) & \xrightarrow[({}_{\mathrm{let}}\nearrow@_0)]{} & \mathbf{let}\ \vec{f}=F(\vec{f})\ \mathbf{in}\ E_0(\vec{f})\,\mathbf{let}\ \vec{g}=G(\vec{g})\ \mathbf{in}\ E_1(\vec{g}) \\
\downarrow{\scriptstyle({}_{\mathrm{let}}\nearrow@_1)} & & \vdots{\scriptstyle({}_{\mathrm{let}}\nearrow@_1)} \\
\mathbf{let}\ \vec{g}=G(\vec{g})\ \mathbf{in}\ (\mathbf{let}\ \vec{f}=F(\vec{f})\ \mathbf{in}\ E_0(\vec{f}))\,E_1(\vec{g}) & & \mathbf{let}\ \vec{f}=F(\vec{f})\ \mathbf{in}\ \mathbf{let}\ \vec{g}=G(\vec{g})\ \mathbf{in}\ E_0(\vec{f})\,E_1(\vec{g}) \\
\vdots{\scriptstyle({}_{\mathrm{let}}\nearrow@_0)} & & \vdots{\scriptstyle(\mathbf{let\text{-}in}_{-\ \mathrm{let}}\nearrow)\cdot=_{\mathrm{ex}}} \\
\mathbf{let}\ \vec{g}=G(\vec{g})\ \mathbf{in}\ \mathbf{let}\ \vec{f}=F(\vec{f})\ \mathbf{in}\ E_0(\vec{f})\,E_1(\vec{g}) & \xrightarrow[(\mathbf{let\text{-}in}_{-\ \mathrm{let}}\nearrow)]{} & \mathbf{let}\ \vec{g}=G(\vec{g}),\ \vec{f}=F(\vec{f})\ \mathbf{in}\ E_0(\vec{f})\,E_1(\vec{g})
\end{array}$$

Then the critical pair theorem for HRSs [6] [10, Thm. 11.6.44] (note that the possibility to find all critical pairs for a HRS is based on a matching algorithm for HRS first described in [6]) yields that $=_{\mathrm{ex}} \cdot {}_{\mathrm{let}}\!\nearrow \cup \to_{\mathrm{ex}}$ is locally confluent. From this, it follows by Lemma 1 that ${}_{\mathrm{let}}\!\nearrow = {}_{\mathrm{let}}\!\nearrow\!/_{=_{\mathrm{ex}}}$ is locally confluent modulo $=_{\mathrm{ex}}$, and that ${}_{[\mathrm{let}]}\!\nearrow$ is locally confluent. □

**Remark 3.** This proof (or actually that of Theorem 6) could also be based on an HRS-analogue of a critical pair theorem by Petersen and Stickel [7, Thm. 9.3] for TRSs that are endowed with an equational theory. Other versions of critical pair theorems for TRSs that are based on 'critical $\to$-pairs modulo $\sim$' (e.g. Jouannaud [5]) suppose that $\to$ is $\sim$-*coherent*: if $t \sim s$ and $t \to^+ t_1$, then there there exist $t_1'$ and $s'$ with $t_1 \twoheadrightarrow t_1'$ and $s \to^+ s'$ such that $t_1' \sim s'$. Yet the relation ${}^{\mathrm{let}}\!\searrow$ here is *not* $=_{\mathrm{ex}}$-coherent: while $\lambda x.\,\mathbf{let}\ f = \lambda y.\, y,\ g = x\ \mathbf{in}\ f\,g$ admits an ${}_{\mathrm{let}}\!\nearrow$-step according to a rule of $({}_{\mathrm{let}}\!\nearrow \lambda)$, the $=_{\mathrm{ex}}$-equivalent term $\lambda x.\,\mathbf{let}\ g = x,\ f = \lambda y.\, y\ \mathbf{in}\ f\,g$ is a ${}_{\mathrm{let}}\!\nearrow$-normal form. In order to apply (an HRS-analogue of) such a theorem, the system has to be extended to one with rewrite relation $=_{\mathrm{ex}} \cdot {}_{\mathrm{let}}\!\nearrow$ by introducing variant rules as in the proof above (also done in [7]).

**Proposition 4.** *${}_{\mathrm{let}}\!\nearrow$ and ${}_{[\mathrm{let}]}\!\nearrow$ are terminating.*

**Proposition 5.** *In every ${}_{\mathrm{let}}\!\nearrow$-normal form, subterms starting with* **let** *occur only at the root or below $\lambda$-abstractions. The same holds for every term representing a ${}_{[\mathrm{let}]}\!\nearrow$-normal form.*

**Theorem 6.** *${}_{[\mathrm{let}]}\!\nearrow$ is confluent and terminating, and has the unique normalization property.*

*Proof.* From Lemma 2 and Proposition 4 by Newman's Lemma [10, Thm. 1.2.1]. □

## 2 Let-sinking

A candidate for a rewrite system for sinking let-bindings is the HRS that arises from the let-lifting HRS $\boldsymbol{R}_{\mathrm{let}}\!\nearrow$ by reversing all of its rules. Unfortunately the resulting system is not confluent. The problem is that the splitting rules for binding-groups, the converses of rules in $(\mathbf{let\text{-}in}_{\_}\ {}_{\mathrm{let}}\!\nearrow)$, allow to sink, for a let-binding-group with two independent parts, each part into the other, so that, in many situations, the results cannot be joined again. We note that adding $(\mathbf{let\text{-}in}_{\_}\ {}_{\mathrm{let}}\!\nearrow)$ would remedy the situation, but at the cost of yielding a non-terminating let-sinking system.

Here we disallow the splitting rules for let-binding-groups altogether, but keep their converses from $(\mathbf{let\text{-}in}_{\_}\ {}_{\mathrm{let}}\!\nearrow)$, yet now call the scheme $({}^{\mathrm{let}}\!\searrow\ \mathbf{let}_{\_})$. Yet we integrate the splitting rules into those let-binding-movement rules for which sinking of entire binding-groups is not always possible, namely rules for sinking let-bindings into the left or right subterm of an application, see the rule schemes $({}_{\mathrm{let}}\!\nearrow @_0)$ and $({}_{\mathrm{let}}\!\nearrow @_1)$ below. As reflected in rules from $({}^{\mathrm{let}}\!\searrow \lambda)$, let-binding-groups can always be sunk into a $\lambda$-abstraction. The rule $(\mathbf{let}_{\_}\ {}^{\mathrm{let}}\!\searrow)$ is the converse of $(\mathbf{let\text{-}in}_{\_}\ {}_{\mathrm{let}}\!\nearrow)$. So we consider the following five rule schemes for sinking let-bindings:

$$({}_{\mathrm{let}}\!\nearrow @_0)\quad \mathbf{let}\ \vec{f} = \vec{F}(\vec{f}),\ \vec{g} = \vec{G}(\vec{f},\vec{g})\ \mathbf{in}\ E_0(\vec{f},\vec{g})\,E_1(\vec{f}) \to \begin{cases} \big(\mathbf{let}\ \vec{g} = \vec{G}(\vec{g})\ \mathbf{in}\ E_0(\vec{g})\big)\,E_1 & \text{if } \vec{f} \text{ is empty} \\ \mathbf{let}\ \vec{f} = \vec{F}(\vec{f})\ \mathbf{in}\ \big(\mathbf{let}\ \vec{g} = \vec{G}(\vec{f},\vec{g})\ \mathbf{in}\ E_0(\vec{f},\vec{g})\big)\,E_1(\vec{f}) & \text{if neither } \vec{f} \text{ nor } \vec{g} \text{ are empty} \end{cases}$$

$$({}_{\mathrm{let}}\!\nearrow @_1)\quad \mathbf{let}\ \vec{f} = \vec{F}(\vec{f}),\ \vec{g} = \vec{G}(\vec{f},\vec{g})\ \mathbf{in}\ E_0(\vec{f})\,E_1(\vec{f},\vec{g}) \to \begin{cases} E_0\,\big(\mathbf{let}\ \vec{g} = \vec{G}(\vec{g})\ \mathbf{in}\ E_1(\vec{g})\big) & \text{if } \vec{f} \text{ is empty} \\ \mathbf{let}\ \vec{f} = \vec{F}(\vec{f})\ \mathbf{in}\ E_0(\vec{f})\,\big(\mathbf{let}\ \vec{g} = \vec{G}(\vec{f},\vec{g})\ \mathbf{in}\ E_1(\vec{f},\vec{g})\big) & \text{if neither } \vec{f} \text{ nor } \vec{g} \text{ are empty} \end{cases}$$

$$({}^{\mathrm{let}}\!\searrow \lambda)\quad \mathbf{let}\ \vec{f} = \vec{F}(\vec{f})\ \mathbf{in}\ \lambda x.\,E(\vec{f},x) \to \lambda x.\,\mathbf{let}\ \vec{f} = \vec{F}(\vec{f})\ \mathbf{in}\ E(\vec{f},x)$$

$$({}^{\text{let}}\!\searrow \textbf{let}_{-}) \quad \textbf{let } \vec{f} = \vec{F}(\vec{f}) \textbf{ in let } \vec{g} = \vec{G}(\vec{f},\vec{g}) \textbf{ in } E(\vec{f},\vec{g}) \;\to\; \textbf{let } \vec{f} = \vec{F}(\vec{f}),\, \vec{g} = \vec{G}(\vec{f},\vec{g}) \textbf{ in } E(\vec{f},\vec{g})$$

$$(\textbf{let}_{-}\,{}^{\text{let}}\!\searrow) \quad \textbf{let } \vec{f} = \vec{F}(\vec{f},g),\, g = G(\vec{f},g,\vec{h}),\, \vec{h} = \vec{H}(\vec{f},g,\vec{h}) \textbf{ in } E(\vec{f},g) \;\to\; \textbf{let } \vec{f} = \vec{F}(\vec{f},g),\, g = \textbf{let } \vec{h} = \vec{H}(\vec{f},g,\vec{h}) \textbf{ in } G(\vec{f},g,\vec{h}) \textbf{ in } E(\vec{f},g)$$

and additionally, the rules of the scheme (exchange) from $\boldsymbol{R}_{\text{let}}{}^{\nearrow}$. By $\boldsymbol{R}^{{}^{\text{let}}\!\searrow}$ we denote the HRS consisting of the five rules above, and by $\boldsymbol{R}^{{}^{\text{let}}\!\searrow^{\text{ex}}}$ its extension with the rule (exchange). The rewrite relations of $\boldsymbol{R}^{{}^{\text{let}}\!\searrow}$ and $\boldsymbol{R}^{{}^{\text{let}}\!\searrow^{\text{ex}}}$ are denoted by ${}^{\text{let}}\!\searrow$ and ${}^{\text{let}}\!\searrow^{\text{ex}}$, respectively.

Since the binding-group merge rules with induced rewrite relation $\to_{\text{merge}}$ are part of both $\boldsymbol{R}_{\text{let}}{}^{\nearrow}$ and $\boldsymbol{R}^{{}^{\text{let}}\!\searrow}$ (in the schemes $(\textbf{let-in}_{-}\,{}_{\text{let}}\!\nearrow)$ in $\boldsymbol{R}_{\text{let}}{}^{\nearrow}$ and $({}^{\text{let}}\!\searrow \textbf{let}_{-})$ in $\boldsymbol{R}^{{}^{\text{let}}\!\searrow}$), the induced let-lifting and let-sinking rewrite relations are not precisely each other's converse. See e.g.:

$$\lambda x.\, \textbf{let } f = x,\, g_1 = g_2\, f,\, g_2 = g_1\, f \textbf{ in } g_1\, g_2 \quad {}^{\text{let}}\!\swarrow\!\searrow\!{}_{\text{let}} \quad \lambda x.\, \textbf{let } f = x \textbf{ in let } g_1 = g_2\, f,\, g_2 = g_1\, f \textbf{ in } g_1\, g_2$$

Observe that the term on the left is a ${}^{\text{let}}\!\searrow$-normal form, and that the ${}_{\text{let}}\!\nwarrow$-step is a $\leftarrow_{\text{merge}}$-step. This example also shows that let-sinking does not always stack let-bindings as deeply as possible. This, however, is consistent with the definition of 'lambda-dropping' and 'block-sinking' in [1].

**Proposition 7.** *Every* ${}^{\text{let}}\!\searrow$*-step is either a* $\to_{\text{merge}}$*-step or the converse of a* ${}_{\text{let}}\!\nearrow$*-step followed by at most one* $\to_{\text{merge}}$*-step. Every* ${}_{\text{let}}\!\nearrow$*-step is either a* $\to_{\text{merge}}$*-step or the converse of a* ${}^{\text{let}}\!\searrow$*-step followed by at most one* $\to_{\text{merge}}$*-step.*

The *let-sinking rewrite relation* ${}^{\text{let}}\!\searrow$ *on* $\boldsymbol{\lambda}_{\mathsf{letrec}}$*-terms* is defined as the rewrite relation ${}^{\text{let}}\!\searrow$ modulo $=_{\text{ex}}$, that is, by: ${}^{\text{let}}\!\searrow \;:=\; {}^{\text{let}}\!\searrow_{/=_{\text{ex}}/} \;=\; =_{\text{ex}} \cdot {}^{\text{let}}\!\searrow \cdot =_{\text{ex}}$. The *let-sinking rewrite relation* ${}^{[\text{let}]}\!\searrow$ *on* $=_{\text{ex}}$*-equivalence classes of* $\boldsymbol{\lambda}_{\mathsf{letrec}}$*-terms* is defined as the class rewrite relation ${}^{[\text{let}]}\!\searrow \;:=\; {}^{\text{let}}\!\searrow_{[=_{\text{ex}}]}$.

As an example we consider the following ${}^{\text{let}}\!\searrow$-rewrite sequence (it is actually a ${}^{\text{let}}\!\searrow$-rewrite sequence) to normal form (this is the converse of the example above for ${}_{\text{let}}\!\nearrow$):

$$\lambda x.\, \textbf{let } f = g,\, g = x \textbf{ in } f\, x \;{}^{\text{let}}\!\searrow\; \lambda x.\, (\textbf{let } f = g,\, g = x \textbf{ in } f)\, x \;{}^{\text{let}}\!\searrow\; \lambda x.\, (\textbf{let } f = \textbf{let } g = x \textbf{ in } g \textbf{ in } f)\, x$$

For similar (trivial) reasons as explained for ${}_{\text{let}}\!\nearrow$, also ${}^{\text{let}}\!\searrow$ is not confluent. But while ${}_{\text{let}}\!\nearrow$ is confluent modulo $=_{\text{ex}}$, this is not the case for ${}^{\text{let}}\!\searrow$, and neither is ${}^{[\text{let}]}\!\searrow$ confluent, yet. In order to see this, consider the following forking ${}^{\text{let}}\!\searrow$-steps:

$$\lambda x.\, \lambda y.\, (\textbf{let } f = \lambda z.\, z \textbf{ in } x)\, y \;\swarrow^{\text{let}}\; \lambda x.\, \lambda y.\, \textbf{let } f = \lambda z.\, z \textbf{ in } x\, y \;{}^{\text{let}}\!\searrow\; \lambda x.\, \lambda y.\, x\, (\textbf{let } f = \lambda z.\, z \textbf{ in } y)$$

Here the $=_{\text{ex}}$-equivalence classes of the reducts (obtained by rules in $({}_{\text{let}}\!\nearrow @_0)$ and $({}_{\text{let}}\!\nearrow @_1)$ respectively) cannot be joined, because the redundant let-binding $f = \lambda z.\, z$ cannot be removed. Therefore we extend the system by two rules for removing redundant and empty let-bindings:

$$\text{(reduce)} \quad \textbf{let } \vec{f} = \vec{F}(\vec{f}),\, \vec{g} = \vec{G}(\vec{f},\vec{g}) \textbf{ in } E(\vec{f}) \;\to\; \textbf{let } \vec{f} = \vec{F}(\vec{f}) \textbf{ in } E(\vec{f})$$

$$\text{(nil)} \quad \textbf{let in } L \;\to\; L$$

which can be called rules for *garbage collection* (in analogy with literature on explicit substitution). The rewrite relation $\to_{\text{gc}}$ is induced by steps according to the rules (reduce) and (nil). The *let-sinking/reduce rewrite relation* ${}^{\text{let}}\!\searrow^{\text{gc}}$ is defined as the rewrite relation ${}^{\text{let}}\!\searrow \cup \to_{\text{gc}}$ modulo $=_{\text{ex}}$, that is, by: ${}^{\text{let}}\!\searrow^{\text{gc}} \;:=\; ({}^{\text{let}}\!\searrow \cup \to_{\text{gc}})_{/=_{\text{ex}}/} \;=\; =_{\text{ex}} \cdot ({}^{\text{let}}\!\searrow \cup \to_{\text{gc}}) \cdot =_{\text{ex}}$ And the *let-sinking/reduce rewrite relation* ${}^{[\text{let}]}\!\searrow^{[\text{gc}]}$ *on* $=_{\text{ex}}$*-equivalence classes of* $\boldsymbol{\lambda}_{\mathsf{letrec}}$*-terms* is defined as the class rewrite relation ${}^{[\text{let}]}\!\searrow^{[\text{gc}]} := {}^{\text{let}}\!\searrow^{\text{gc}}{}_{[=_{\text{ex}}]}$.

Using these relations we can join the forking steps from above as follows:

$$\lambda x.\, \lambda y.\, (\textbf{let } f = \lambda z.\, z \textbf{ in } x)\, y \;\twoheadrightarrow_{\text{gc}}\; \lambda x.\, \lambda y.\, x\, y \;\twoheadleftarrow_{\text{gc}}\; \lambda x.\, \lambda y.\, x\, (\textbf{let } f = \lambda z.\, z \textbf{ in } y)$$

**Remark 8.** In [2, 9] we introduce and study a rewrite system (formalized as a Combinatory Reduction System) for unfolding $\boldsymbol{\lambda}_{\mathsf{letrec}}$-terms into infinite $\lambda$-terms. This system contains a rule scheme that enables more general steps than those of the scheme (reduce), namely:

$$(\varrho_{\triangledown}^{\text{reduce}}): \quad \textbf{letrec}\ f_1 = L_1 \ldots f_n = L_n\ \mathsf{in}\ L \ \to\ \textbf{letrec}\ f_{j_1} = L_{j_1} \ldots f_{j_{n'}} = L_{j_{n'}}\ \mathsf{in}\ L$$
$$(\text{if } f_{j_1}, \ldots, f_{j_{n'}} \text{ are the recursion variables that are reachable from } L)$$

However, due to the presence of the rule scheme (exchange) in the systems we consider here, every step according to a rule of $(\varrho_{\triangledown}^{\text{reduce}})$ can be simulated by a number of $\to_{\text{ex}}$-steps followed by a step according to a rule of (reduce). Thus the syntactically easier rules of (reduce) suffice here. The availability of the rules of (exchange) also enables the use of the rules $({}_{\text{let}}\!\nearrow \lambda)$ and $({}^{\text{let}}\!\searrow @_i)$ $(i \in \{0,1\})$ in which a call graph analysis is enforced by a pattern of rather easy form.

**Lemma 9.** ${}^{\text{let}}\!\searrow^{\text{gc}}$ *is locally confluent modulo* $=_{\text{ex}}$*, and* ${}^{[\text{let}]}\!\searrow^{[\text{gc}]}$ *is locally confluent.*

*Proof (Idea).* Similarly as in the proof of Lemma 2, a critical-pair analysis is carried out for a HRS $\boldsymbol{R}^{\text{let}\searrow\text{gc}}$ with $\to_{\text{ex}} \cup\ =_{\text{ex}} \cdot ({}^{\text{let}}\!\searrow \cup \to_{\text{gc}})$ as its rewrite relation. Here the analysis is more laborious (two more rules), and considerably more tedious (for three schemes, $({}_{\text{let}}\!\nearrow @_0)$, $({}_{\text{let}}\!\nearrow @_1)$, and $(\textbf{let}\_\,{}^{\text{let}}\!\searrow)$, the rule patterns create splits of let-binding-groups, which in order to join critical steps requires a careful analysis of the possible call graphs between let-bindings in their source term). The lemma folows by the Critical Pair Theorem of [6] and Lemma 1. □

**Proposition 10.** ${}^{\text{let}}\!\searrow^{\text{gc}}$ *and* ${}^{[\text{let}]}\!\searrow^{[\text{gc}]}$ *are terminating.*

**Theorem 11.** ${}^{[\text{let}]}\!\searrow^{[\text{gc}]}$ *is confluent, terminating, and has the unique normalization property.*

The properties stated for ${}^{[\text{let}]}\!\searrow^{[\text{gc}]}$ in Thm. 11 and for ${}_{[\text{let}]}\!\nearrow$ in Thm. 6 can also be shown for the extension ${}^{[\text{let}]}\!\nearrow^{[\text{gc}]}$ of the let-lifting rewrite relation ${}_{[\text{let}]}\!\nearrow$ by incorporating $\to_{\text{gc}}$-steps. Finally, a comprehensive HRS for let-floating in both upward and downward direction, and for reducing binding-groups can be obtained by gathering all rules underlying ${}_{\text{let}}\!\nearrow$ and ${}^{\text{let}}\!\searrow^{\text{gc}}$.

*Acknowledgement.* We want to thank the reviewers for their valuable comments and suggestions.

# References

[1] Olivier Danvy and Ulrik P. Schultz. Lambda-dropping: transforming recursive equations into programs with block structure. *Theoretical Computer Science*, 248(1-2):243 – 287, 2000. PEPM'97.

[2] Clemens Grabmayer and Jan Rochel. Expressibility in the Lambda-Calculus with Letrec. Technical Report arXiv:1208.2383, `arxiv.org`, August 2012. `http://arxiv.org/abs/1208.2383`.

[3] Clemens Grabmayer and Jan Rochel. Term Graph Representations for Cyclic Lambda Terms. In *Proc. of TERMGRAPH 2013*, number 110 in EPTCS, 2013. `http://arxiv.org/abs/1302.6338v1`.

[4] Simon Peyton Jones. *The Implementation of Functional Progr. Languages.* Prentice-Hall, 1987.

[5] Jean-Pierre Jouannaud. Confluent and coherent equational term rewriting systems application to proofs in abstract data types. In Giorgio Ausiello and Marco Protasi, editors, *CAAP'83*, volume 159 of *Lecture Notes in Computer Science*, pages 269–283. Springer Berlin Heidelberg, 1983.

[6] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, 1998.

[7] Gerald E. Peterson and Mark E. Stickel. Complete Sets of Reductions for Some Equational Theories. *JACM*, 28(2):233–264, 1981.

[8] Simon Peyton Jones, Will Partain, and André Santos. Let-floating: moving bindings to give faster programs. In *Proceedings of the first ACM SIGPLAN international conference on Functional programming*, ICFP '96, pages 1–12, New York, NY, USA, 1996. ACM.

[9] Jan Rochel and Clemens Grabmayer. Confluent unfolding in the $\lambda$-calculus with letrec. In *Proceedings of IWC 2013 ($2^{nd}$ International Workshop on Confluence)*, 2013.

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

# Author Index