IWC 2012

1st International Workshop on Confluence

Proceedings

Editors: Nao Hirokawa & Aart Middeldorp

May 29, 2012, Nagoya, Japan

Preface

This report contains the proceedings of the *1st International Workshop on Confluence (IWC 2012)*. The workshop was held in Nagoya on May 29, 2012 and collocated with the 23rd International Conference on Rewriting Techniques and Applications (RTA 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 2012 received 6 submissions. Each submission was reviewed by 3 program committee members. After careful deliberations the program committee decided to accept all submissions, which are contained in this report. Also included are (extended) abstracts of the invited presentations by *Yoshihito Toyama* on *Type Introduction for Confluence Proofs* and *Vincent van Oostrom* on *Decreasing Proof Orders — Interpreting Conversions in Involutive Monoids*. Furthermore, this report contains short descriptions of the four tools that participated in the 1st Confluence Competition (CoCo 2012). This competition ran live during the workshop and the results are available at `http://coco.nue.riec.tohoku.ac.jp/2012/`.

Several persons helped to make IWC 2012 a success. We are grateful to the members of the program committee and to Naoki Nishida and Masahiko Sakai of the organisation committee of RTA 2012 for their work. A special thanks to Harald Zankl who made the live run of CoCo possible.

Innsbruck & Kanazawa, May 2012 *Nao Hirokawa & Aart Middeldorp*

Conference Organisation

Organising Committee

Nao Hirokawa	*Japan Advanced Institute of Science and Technology*	
Aart Middeldorp	*University of Innsbruck*	
Naoki Nishida	*Nagoya University*	

Program Committee

Takahito Aoto	*University of Tohoku*	
Nao Hirokawa	*Japan Advanced Institute of Science and Technology*	(co-chair)
Aart Middeldorp	*University of Innsbruck*	(co-chair)
Femke van Raamsdonk	*VU University Amsterdam*	
Aaron Stump	*The University of Iowa*	
Rakesh M. Verma	*University of Houston*	

Table of Contents

Invited Papers

Contributed Papers

Tool Descriptions

Decreasing Proof Orders*

Interpreting Conversions in Involutive Monoids

Vincent van Oostrom
Janskerkhof 13, 3512 BL Utrecht, Netherlands
Utrecht University, Department of Philosophy
Vincent.vanOostrom@phil.uu.nl

Abstract

We introduce the *decreasing* proof order. It orders a conversion above another conversion if the latter is obtained by filling any peak in the former by a decreasing diagram. The result is developed in the setting of involutive monoids.

Definition 1. An *involutive monoid* is a set endowed with an associative binary operation (c) with identity element (e) and involutive anti-automorphism (i).

Examples 2. For a set of labels L, the strings over *grave* labels $\grave{\ell}$ and *acute* labels $\acute{\ell}$ with juxtaposition, mirroring (swapping marks), and empty string ε, give an involutive monoid $\widehat{L}$. (We use $\hat{\ell}$ to denote $\acute{\ell}$ or $\grave{\ell}$.) Setting $i(a) \mapsto a$ turns any commutative monoid into an involutive one, e.g. the free commutative monoid $[L]$ over L with multiset sum $\uplus$ and empty multiset $[]$. Natural number triples (n,m,k) with $c((n_1,m_1,k_1),(n_2,m_2,k_2)) = (n_1+n_2, m_1+k_1\cdot n_2+m_2, k_1+k_2)$, $i((n,m,k)) = (k,m,n)$, and $e = (0,0,0)$, give an involutive monoid $\mathcal{A}$.

$\widehat{L}$ is the free involutive monoid over L and its elements correspond bijectively to the closed normal forms over L of the term rewrite system (having unique normal forms) obtained by completing the involutive monoid axioms into rules:

$$\begin{aligned} c(c(x,y),z) &\to c(x,c(y,z)) \\ c(x,e) &\to x \\ c(e,x) &\to x \\ i(i(x)) &\to x \\ i(c(x,y)) &\to c(i(y),i(x)) \\ i(e) &\to e \end{aligned}$$

Definition 3. An involutive monoid is *well-founded* if it comes equipped with a well-founded partial order, i.e. a transitive and terminating relation.

Examples 4. Let $\succ$ be a well-founded partial order on L, The multiset extension $\succ_{mul}$ of $\succ$ makes $[L]$ a well-founded involutive monoid. Comparing triples by means of $>$ on their second component, turns $\mathcal{A}$ into well-founded involutive monoid. Terms over the signature $\widehat{L}$, assigning arity zero to ε and other strings their length plus one, may be *flattened* to strings in $\widehat{L}$ by the obvious inorder traversal map $\flat$. Conversely, splitting on $\succ$-maxima *stratifies* $\widehat{L}$ into terms:

$$\begin{aligned} \varepsilon^\sharp &= \varepsilon \\ (s_0\hat{\ell}_1\ldots\hat{\ell}_n s_n)^\sharp &= \hat{\ell}_1\ldots\hat{\ell}_n(s_0^\sharp,\ldots,s_n^\sharp) \end{aligned}$$

with $\ell_1,\ldots,\ell_n$ all occurrences of labels $\succ$-maximal within the string. The set $\underline{L}$ of terms on which $\sharp$ and $\flat$ are each other's inverse,[1] gives an involutive monoid by defining its operations

*This is an extended abstract of a paper with the same title accepted for publication in TCS.

[1] Their strings have pairwise $\succ$-incomparable labels and are related by the $<$-Hoare order.

via those on $\widehat{L}$. It is made well-founded [2, Theorem 1]by the iterative lexicographic path order $\succ_{ilpo}$ [2, Definition 2] induced by the relation $\succ$ on the signature. The relation $\succ$ is defined as the lexicographic product of $>_{mul}$ and $>$, interpreting a string as the pair consisting of its multiset of labels and its *area*, the image of the homomorphic extension of $\grave{\ell} \mapsto (1,0,0)$, illustrated in Figure 1 for strings in $\widehat{\{*\}}$ writing $\backslash$ for $\grave{*}$ and $/$ for $\acute{*}$. Moreover, argument places are ordered

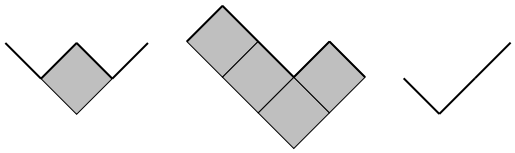


Figure 1: $\backslash/\backslash/$ has area 1, $/\backslash\backslash/\backslash$ has area 4, and $\backslash//$ has area 0

lexicographically by choosing an arbitrary but fixed total order on them compatible with the marks: if the $i+1^{st}$ label has a grave (acute) mark, then the i^{th} argument place comes before (after) the $i+1^{st}$.

Theorem 5. *If $\succ$ is a well-founded partial order on L, then $\Rightarrow^+$ makes $\widehat{L}$ a well-founded involutive monoid, where the* decreasing *rewrite relation $\Rightarrow$ on $\widehat{L}$, is generated by all rewrite rules on $\widehat{L}$ of shape (in EBNF):*

$$\acute{\ell}\grave{m} \quad \Rightarrow \quad \{(\ell>)\}[\grave{m}]\{(\ell,m>)\}[\acute{\ell}]\{(m>)\}$$

with $(m>)$ the set comprising all strings over marked labels $\hat{\kappa}$ with $m \succ \kappa$.

Proof. As $\Rightarrow^+$ is transitive by definition, it remains to show termination. It suffices to show that $s_1\acute{\ell}\grave{m}s_2 \Rightarrow s_1ss_2$ entails $(s_1\acute{\ell}\grave{m}s_2)^\sharp \succ_{ilpo} (s_1ss_2)^\sharp$, if s is of shape $\{(\ell>)\}[\grave{m}]\{(\ell,m>)\}[\acute{\ell}]\{(m>)\}$. The proof is by induction on the length of the string distinguishing cases on whether ℓ, m are $\succ$-maximal in $s_1\acute{\ell}\grave{m}s_2$.

- if both ℓ,m are $\succ$-maximal then the head of $(s_1\acute{\ell}\grave{m}s_2)^\sharp$ is $\succ$-related to the head of $(s_1ss_2)^\sharp$; by the shape of the $\Rightarrow$-rules either its multiset of labels $>_{mul}$-decreases, or it stays the same and its area decreases. By the footnote the head is $\succ$-related to all function symbols in $(s_1ss_2)^\sharp$;
- if neither ℓ nor m is $\succ$-maximal then the heads of $(s_1\acute{\ell}\grave{m}s_2)^\sharp$ and $(s_1ss_2)^\sharp$ as well as all non-$\succ$-maximal substrings other than the ones affected by the $\Rightarrow$-step, are the same. We conclude by the induction hypothesis for the affected substrings and closure under contexts of $\succ_{ilpo}$;
- if one of the labels is $\succ$-maximal, say w.l.o.g. the second m, and the other, ℓ, is not, then if the multiset of labels $>_{mul}$-decreases we conclude as in the first item. Otherwise, the multiset stays the same, but the interpretation of the substring *before* the occurrence of $\grave{m}$ in rewritten substrings, $\succ_{ilpo}$-decreases as the $\Rightarrow$-rule enforces replacing $\acute{\ell}$ by a string with labels in $(\ell>)$. The $\Rightarrow$-rule moreover enforces that the substring *after* the occurrence of $\grave{m}$ still only comprises labels to which some label in the head $\succ$-relates. □

Example 6. Setting $m \succ \ell, \kappa$, we have the decreasing $\Rightarrow$-rewrite sequence, where we have underlined in each step the substring being replaced:

$$\underline{\acute{m}\grave{\kappa}\acute{\ell}\grave{m}} \Rightarrow_① \acute{\ell}\acute{m}\underline{\acute{\ell}\grave{m}} \Rightarrow_② \acute{\ell}\underline{\acute{m}\grave{m}\grave{\ell}} \Rightarrow_③ \underline{\acute{\ell}\grave{\ell}}\grave{m}\acute{m}\grave{\ell} \Rightarrow_④ \acute{\ell}\grave{m}\underline{\acute{m}\grave{\ell}} \Rightarrow_⑤ \underline{\acute{\ell}\grave{m}\acute{\kappa}\acute{\ell}} \Rightarrow_⑥ \grave{m}\acute{\kappa}\acute{\ell}$$

Stratifying this $\Rightarrow$-sequence by means of $\sharp$ yields the $\succ_{ilpo}$-sequence:

$$
\begin{array}{rl}
\acute{m}\grave{m}(\varepsilon,\grave{\kappa}\acute{\ell}(\varepsilon,\varepsilon,\varepsilon),\varepsilon) & \\
\quad \forall_{ilpo}\,① & \text{decrease of multiset at position 1} \\
\acute{m}\grave{m}(\acute{\ell}(\varepsilon,\varepsilon),\acute{\ell}(\varepsilon,\varepsilon),\varepsilon) & \\
\quad \forall_{ilpo}\,② & \text{decrease of multiset at position 1} \\
\acute{m}\grave{m}(\acute{\ell}(\varepsilon,\varepsilon),\varepsilon,\grave{\ell}\,(\varepsilon,\varepsilon)) & \\
\quad \forall_{ilpo}\,③ & \text{decrease of area at the root} \\
\grave{m}\acute{m}(\acute{\ell}\grave{\ell}\,(\varepsilon,\varepsilon,\varepsilon),\varepsilon,\grave{\ell}\,(\varepsilon,\varepsilon)) & \\
\quad \forall_{ilpo}\,④ & \text{decrease of multiset at position 0} \\
\grave{m}\acute{m}(\acute{\ell}(\varepsilon,\varepsilon),\varepsilon,\grave{\ell}\,(\varepsilon,\varepsilon)) & \\
\quad \forall_{ilpo}\,⑤ & \text{decrease of multiset at the root} \\
\grave{m}(\acute{\ell}(\varepsilon,\varepsilon),\acute{\kappa}\acute{\ell}(\varepsilon,\varepsilon,\varepsilon)) & \\
\quad \forall_{ilpo}\,⑥ & \text{decrease of multiset at position 0} \\
\grave{m}(\varepsilon,\acute{\kappa}\acute{\ell}(\varepsilon,\varepsilon,\varepsilon)) &
\end{array}
$$

Note that in the $\succ_{ilpo}$①-step the subterm at position 0 *increases*, but this is not harmful since it comes lexicographically *after* the subterm at position 1.

Definition 7. A family $(\to_\ell)_{\ell\in L}$ of rewrite relations is *decreasing* if its set L of labels comes equipped with a well-founded partial order $\succ$. A rewrite relation $\to$ is *decreasing* if $\to = \bigcup_{\ell\in L} \to_\ell$ for some family of rewrite relations $(\to_\ell)_{\ell\in L}$ that is decreasing with respect to some $\succ$, and every pair of consecutive steps $a \leftarrow_\ell c \to_m b$ can be replaced by a conversion between a and b such that mapping the conversions to $\widehat{L}$-strings by the maps $\to_\kappa \mapsto \grave{\kappa}$ and $\leftarrow_\kappa \mapsto \acute{\kappa}$ induces a $\Rightarrow$-step.

Viewing conversions as proofs in equational logic one may think of $\Rightarrow$ as inducing an order on proofs which, in view of the above, we dub a *decreasing* proof order.

Corollary 8 ([3, Theorem 3])**.** *A decreasing rewrite relation is confluent.*

Proof. Given a conversion, choose any L-labelling of it. As long as it is not yet a valley, it can be transformed by the decreasingness assumption. By definition this induces a $\Rightarrow$-sequence. That ends by the theorem yielding a valley. □

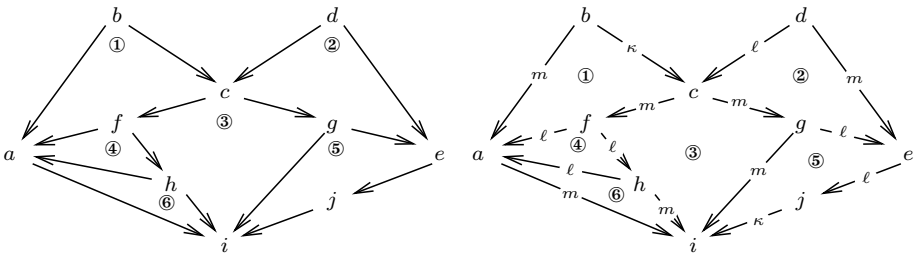


Figure 2: Decreasingness of rewrite relation

Example 9. The rewrite relation on the left in Figure 2 is decreasing hence confluent, as verified by the rewrite family on its right, and Example 6.

On [3, p. 315] we wrote:

> ... one could be led to believe the decreasing diagrams technique is just the 'sum of Newman's Lemma and the Lemma of Hindley–Rosen'. The following examples show it is much more powerful ...

Extant proofs [3, 1] shed no light on the dependencies between the two results. The present proof combines their essential ingredients, the multiset of labels respectively the area of a string, by means of the iterative lexicographic path order, and specialises to either by forgetting the other ingredient.

References

[1] Jean-Pierre Jouannaud and Vincent van Oostrom. Diagrammatic confluence and completion. In *36th International Colloquium on Automata, Languages and Programming, ICALP 2009, Rhodes, Greece, July 2009, Proceedings*, volume 5556 of *Lecture Notes in Computer Science*, pages 212–222. Springer, 2009. (doi:10.1007/978-3-642-02930-1_18).

[2] Jan Willem Klop, Vincent van Oostrom, and Roel de Vrijer. Iterative lexicographic path orders. In *Algebra, Meaning and Computation: Essays dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, volume 4060 of *Lecture Notes in Computer Science*, pages 541–554, 2006. (doi:10.1007/11780274_28).

[3] Vincent van Oostrom. Confluence by decreasing diagrams, converted. In Andrei Voronkov, editor, *Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 2007. Proceedings*, volume 5117 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2008. (doi:10.1007/978-3-540-70590-1_21).

Type Introduction for Confluence Proofs

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

The confluence property is one of the most important properties of term rewriting systems, and various criteria for proving this property have been widely investigated. A necessary and sufficient criterion for confluence of terminating term rewriting systems was demonstrated by Knuth and Bendix (1970). For non-terminating term rewriting systems, various sufficient criteria for confluence of left-linear systems were presented by Rosen (1973), Huet (1980), Toyama (1988), and van Oostrom (1997). However, few criteria have been proposed for confluence of term rewriting systems that are non-left-linear and non-terminating. Thus, it is still worthwhile studying criteria for these systems.

A powerful technique for showing confluence of non-left-linear non-terminating term rewriting systems is a divide-and-conquer method by type introduction. The confluence property is persistent, that is, for any term rewriting system $\mathcal{R}$, $\mathcal{R}$ is confluent if and only if its typed system $\mathcal{R}^\tau$ is confluent (Aoto and Toyama 1997). Thus, persistency of confluence guarantees that if a term rewriting system $\mathcal{R}$ is decomposed into small subsystems by introducing an appropriate type and each of them is confluent then R is confluent. This divide-and-conquer method is successfully used in the automated confluence prover ACP developed by Aoto et al. (2009). However, if $\mathcal{R}$ is not decomposed into small subsystems by introducing types, this method does not work.

A new technique for confluence proof based on persistency is recently developed by Suzuki, Aoto and Toyama (2011), which can be applied to term rewriting systems that are not decomposed by introducing types. The idea is that the classification of terms by introducing types aids in analyzing reduction sequences induced by rewrite rules. For instance, if the non-left-linear variables of a rewrite rule has type σ and every term with type σ is terminating then under some assumption these non-left-linear variables in the rewrite rule can be replaced with the normal terms with type σ. This replacement transforms a non-left-linear rule into an infinite number of left-linear rules, and the non-left-linear system is confluent if the obtained infinite left-linear system is confluent. In this talk we will illustrate these type introduction techniques for proving confluence of term rewriting systems.

A Proof Order for Decreasing Diagrams*

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

Abstract

In this note we describe a well-founded proof order that entails the decreasing diagrams technique, i.e., it orders peaks of locally decreasing diagrams above their joining sequences. We also investigate an extension that promises to be useful for proving confluence modulo.

Unrelated to this proof order we also present an example showing that witnesses for non-confluence can not always be found by starting from critical pairs alone, even for linear TRSs.

1 Introduction

In this note we revisit the decreasing diagrams technique [8] for proving confluence. Our confluence proof is based on a proof order, which we use to prove termination of the *proof transformation system* defined by the locally decreasing diagrams. A similar approach is used in the correctness proof for completion by Bachmair and Dershowitz [2]. The distinguishing property of a proof order is that proof concatenation becomes a monotone operation, so that admissibility of a proof transformation $P \Rightarrow Q$ can be established by comparing P to Q. This simplifies finding new proof transformations.

This work is also inspired by [6], where Jouannaud and van Oostrom define a well-founded order on proofs in order to establish that local decreasingness implies confluence. However, it is not monotone for concatenation, and therefore not a proof order. Hence they have to consider whole proofs when showing that eliminating a local peak by a locally decreasing diagram results in a decrease in the proof measure. Another influence comes from Aoto and Toyama [1], who prove an abstract lemma for confluence modulo by proof rewriting arguments.

The remainder of this paper is structured as follows: In Section 2, we introduce our proof order abstracted to *proof strings*. We use this order to re-prove validity of the decreasing diagrams technique in Section 3. In Section 4, we explore admissible proof transformations that are helpful for proving confluence modulo, leading to a generalization of a result by Ohlebusch [7].

In Section 5 we depart from the topic of proof orders and proving confluence and turn to establishing non-confluence. An obvious idea for finding counterexamples to confluence is to start with critical peaks and try to find reducts which are not joinable. It is well-known that this is not sufficient for non-left-linear TRSs (Huet, [5]). We give an example that shows that this approach is insufficient even for linear TRSs.

1.1 Preliminaries

For the whole paper, we fix a set of labels $\mathcal{L}$ equipped with a well-founded order $\succ$. Given labels $\alpha, \gamma \in \mathcal{L}$, we define $\Upsilon\alpha = \{\gamma \in \mathcal{L} \mid \alpha \succ \gamma\}$ and $\Upsilon\alpha, \gamma = \Upsilon\alpha \cup \Upsilon\gamma$.

We use $\to$ ($\mapsto$) to denote (symmetric) rewrite relations, with the usual convention that $\leftarrow$, $\leftrightarrow$, $\xrightarrow{=}$, $\xrightarrow{*}$ denote the inverse, symmetric closure, reflexive closure and reflexive, transitive closure of $\to$, respectively. Given a family $(\underset{\alpha}{\to})_{\alpha\in\mathcal{L}}$ and $S \subseteq \mathcal{L}$, we let $\underset{S}{\to} = \bigcup_{\alpha\in S} \underset{\alpha}{\to}$.

The lexicographic product of strict partial orders $>_1$ and $>_2$ is denoted by $>_1 \times_{\text{lex}} >_2$, while $>_{\text{mul}}$ is the multiset extension of $>$. We use $\{\!|$ and $|\!\}$ as brackets for multisets, e.g., $\{\!|\alpha, \alpha, \gamma|\!\}$.

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

2 Ordering Proof Strings

This section is devoted to developing our proof order in an abstract setting, where instead of rewrite proofs we consider only proof strings, abstracting from the objects of abstract rewrite systems. The resulting *proof string order* is defined in two stages. First we map proof strings to nested multisets of pairs. Then we compare these multisets by an order related to the recursive path order.

Definition 1. We introduce proof strings, an abstract notation for proofs.

- A *(proof) step* is either a *left step* $\overleftarrow{\alpha}$ or a *right step* $\overrightarrow{\alpha}$, where $\alpha \in \mathcal{L}$. We define $\overleftrightarrow{S} = \{\overleftarrow{\alpha}, \overrightarrow{\alpha} \mid \alpha \in S\}$ for $S \subseteq \mathcal{L}$ and use $\overleftrightarrow{\alpha}$ for variables ranging over steps ($\overleftrightarrow{\alpha} \in \overleftrightarrow{\mathcal{L}}$), and α for the corresponding labels. We lift $\succ$ to steps by letting $\overleftrightarrow{\alpha} \succ \overleftrightarrow{\gamma}$ iff $\alpha \succ \gamma$.
- A *(proof) string* is a sequence of steps. The set of all proof strings is denoted by $\mathcal{P}$, and $\cdot$ is the concatenation operation on strings. The empty proof is ϵ.
- The *inverse* $(\overleftrightarrow{\alpha})^{-1}$ of a step $\overleftrightarrow{\alpha}$ is defined by $(\overrightarrow{\alpha})^{-1} = \overleftarrow{\alpha}$ and $(\overleftarrow{\alpha})^{-1} = \overrightarrow{\alpha}$. This operation extends to proof strings by $\epsilon^{-1} = \epsilon$, $(P \cdot \overleftrightarrow{\alpha})^{-1} = (\overleftrightarrow{\alpha})^{-1} \cdot P^{-1}$.

Remark 1. Together with inverse and concatenation, proof strings form an involutive monoid (van Oostrom, IWC 2012).

Definition 2. A well-founded order $\gg$ on strings is a *proof string order* if string concatenation and inverse are strictly monotone, i.e., $P \gg Q$ implies $P \cdot R \gg Q \cdot R$, $R \cdot P \gg R \cdot Q$ and $P^{-1} \gg Q^{-1}$.

Next we show how proof strings are mapped to nested multisets.

Definition 3. We define operations $[\cdot]^l$, $[\cdot]^r$ and $[\cdot]^m$, mapping proof strings to (nested) multisets of pairs of steps and transformed strings, inductively as follows:

- $[\epsilon]^l = \varnothing$, $[\overleftarrow{\alpha} \cdot P]^l = \{\!\{(\overleftarrow{\alpha}, [P]^m)\}\!\} \cup [P]^l$, $[\overrightarrow{\alpha} \cdot P]^l = [P]^l$, collecting left steps and the transformations of the substring following each left step into a multiset.
- $[\epsilon]^r = \varnothing$, $[P \cdot \overrightarrow{\alpha}]^r = \{\!\{(\overrightarrow{\alpha}, [P]^m)\}\!\} \cup [P]^r$, $[P \cdot \overleftarrow{\alpha}]^r = [P]^r$, collecting right steps and the transformations of their preceding substrings.
- $[P]^m = [P]^l \cup [P]^r$.

Example 2. We have $[\overleftarrow{\alpha} \cdot \overrightarrow{\gamma}]^m = \{\!\{(\overleftarrow{\alpha}, [\overrightarrow{\gamma}]^m), (\overrightarrow{\gamma}, [\overleftarrow{\alpha}]^m)\}\!\} = \{\!\{(\overleftarrow{\alpha}, \{\!\{(\overrightarrow{\gamma}, \varnothing)\}\!\}), \{\!\{(\overrightarrow{\gamma}, \{\!\{(\overleftarrow{\alpha}, \varnothing)\}\!\})\}\!\}$.

The result of the transformation $[\cdot]^m$ grows exponentially in the string length, but it is highly redundant: each multiset occurring in $[P]^m$ corresponds to a substring of P.

Lemma 4. *The definition of* $[P]^m$ *is symmetric. Formally, we have* $[P^{-1}]^m = ([P]^m)^{-1}$ *and* $[P^{-1}]^l = ([P]^r)^{-1}$*, where the inverse on nested multisets is defined recursively by*

$$s^{-1} = \{\!\{((\overleftrightarrow{\alpha})^{-1}, t^{-1}) \mid (\overleftrightarrow{\alpha}, t) \in s\}\!\}$$

Proof. By induction on the length of P. □

Definition 5. We order these multisets by $\succ_\oplus$, defined inductively: $s \succ_\oplus \{\!\{(\overleftrightarrow{\gamma_1}, t_1), \ldots, (\overleftrightarrow{\gamma_m}, t_m)\}\!\}$ if $s \succ_\oplus t_j$ for $1 \le j \le m$ and $s \gg_{\text{mul}} t$, where $\gg = \succ \times_{\text{lex}} \succ_\oplus$. Furthermore, we define the proof string order $\succ_\bullet$ as follows:

$$P \succ_\bullet Q \quad \text{iff} \quad [P]^m \succ_\oplus [Q]^m$$

Lemma 6. *The relations $\succ_\oplus$ and $\succ_\bullet$ are well-founded partial orders.*

Proof. We only have to establish that $\succ_\oplus$ is a well-founded order. First we show transitivity, i.e., that $s \succ_\oplus t \succ_\oplus u$ implies $s \succ_\oplus u$, by induction on u. Assume that $u = \{\!\{(\overleftrightarrow{\eta_1}, u_1), \ldots, (\overleftrightarrow{\eta_n}, u_n)\}\!\}$. Now from $t \succ_\oplus u$ follows $t \succ_\oplus u_i$, hence $s \succ_\oplus u_i$ by the induction hypothesis. Furthermore with $\gg = \succ \times_{\text{lex}} \succ_\oplus$, we have $s \gg_{\text{mul}} t \gg_{\text{mul}} u$, and therefore $s \gg_{\text{mul}} u$ by the induction hypothesis, because the transitivity proofs for $\times_{\text{lex}}$ and $\cdot_{\text{mul}}$ only rely on the transitivity for elements actually present in the pairs respectively multisets. Hence $s \succ_\oplus u$, as claimed.

Now it remains to show that $\succ_\oplus$ is well-founded. In order to do that, we exhibit a relation between $\succ_\oplus$ and the recursive path order with custom status as introduced by Ferreira [3]. To this end, we define a signature $\mathcal{F}$, a mapping $[\cdot]^t$ of nested multisets to ground terms over $\mathcal{F}$ and a lifting Λ on relations between terms over $\mathcal{F}$ as follows:

- $\mathcal{F} = \mathcal{P}$, where each proof string in $\mathcal{F}$ has its length as arity.
- $[\{\!\{(\overleftrightarrow{\alpha_1}, s_1), \ldots, (\overleftrightarrow{\alpha_n}, s_n)\}\!\}]^t = \overleftrightarrow{\alpha_1} \ldots \overleftrightarrow{\alpha_n}([s_1]^t, \ldots, [s_n]^t)$, using any order of the multiset elements.
- $s >^\Lambda t$, if and only if $s^\lambda \gg_{\text{mul}} t^\lambda$, where $[\overleftrightarrow{\alpha_1} \ldots \overleftrightarrow{\alpha_n}(s_1, \ldots, s_n)]^\lambda = \{\!\{(\overleftrightarrow{\alpha_1}, s_1), \ldots, (\overleftrightarrow{\alpha_n}, s_n)\}\!\}$ and $\gg = \succ \times_{\text{lex}} >$.

It is easy to see that Λ is a *term lifting* [3, Definition 3.16] and also a *status* [3, Definition 4.7]. Let $\gg_\bullet$ be the rpo defined by this status, namely: $s \gg_\bullet t$ iff $s = \overleftrightarrow{\alpha_1} \ldots \overleftrightarrow{\alpha_n}(s_1, \ldots, s_n)$, $t = \overleftrightarrow{\gamma_1} \ldots \overleftrightarrow{\gamma_m}(t_1, \ldots, t_m)$ and

1. $s_i = t$ or $s_i \gg_\bullet t$ for some $1 \le i \le n$, or
2. $s \gg_\bullet t_j$ for $1 \le j \le m$ and $s \gg_\bullet^\Lambda t$.

By the properties of rpo [3, Theorem 4.19], $\gg_\bullet$ is well-founded. We conclude that $\succ_\oplus$ is well-founded by noting that $[P]^m \succ_\oplus [Q]^m$ implies $[[P]^m]^t \gg_\bullet [[Q]^m]^t$, which can be shown by unfolding the definitions of $\succ_\oplus$, $[\cdot]^t$ and $\gg_\bullet$, using only the second case in the definition of $\gg_\bullet$. Note in particular that each application $[\cdot]^\lambda$ reverses one level of the transformation $[\cdot]^t$. □

Remark 3. It would be nice to avoid the complications of using a general status for rpo. However, both the lexicographic and the multiset comparisons are essential, and splitting the signature $\mathcal{F}$ into several levels conflicts with the requirement that if $s \gg f(t_1, \ldots, t_n)$, then $s \gg t_i$ for all i.

Theorem 7. *The order $\succ_\bullet$ is a proof string order.*

Proof. By Lemma 6, $\succ_\bullet$ is a well-founded order. To see that string inverse is monotone with respect to $\succ_\bullet$, apply Lemma 4 and observe that $\succ_\oplus$ is invariant under inversion of steps.

For concatenation we first prove that it is monotone in the second argument, i.e., $Q \succ_\bullet Q'$ implies $P \cdot Q \succ_\bullet P \cdot Q'$. We proceed by induction on the length of P. If $P = \epsilon$, then there is nothing to prove. Otherwise, there are two cases to consider, $P = P' \cdot \overleftarrow{\alpha}$ and $P = P' \cdot \overrightarrow{\alpha}$. Their proofs are very similar, so we only handle the first case here.

Assume that $P = P' \cdot \overleftarrow{\alpha}$. We let $\gg = \succ \times_{\text{lex}} \succ_\oplus$. We prove by induction on Q that $[\overleftarrow{\alpha} \cdot Q]^m \gg_{\text{mul}} [\overleftarrow{\alpha} \cdot Q']^m$. Unfolding one level of $[\cdot]^m$, we can relate $[\overleftarrow{\alpha} \cdot Q]^m$ to $[Q]^m$:

$$[\overleftarrow{\alpha} \cdot Q]^m = \{\!\{(\overleftarrow{\alpha}, [Q]^m)\}\!\} \cup \{\!\{(\overleftarrow{\gamma}, [R]^m) \mid (\overleftarrow{\gamma}, [R]^m) \in [Q]^m\}\!\} \cup \{\!\{(\overrightarrow{\gamma}, [\overleftarrow{\alpha} \cdot R]^m) \mid (\overrightarrow{\gamma}, [R]^m) \in [Q]^m\}\!\}.$$

We say that $(\overleftarrow{\gamma}, [R]^m)$, $(\overrightarrow{\gamma}, [\overleftarrow{\alpha} \cdot R]^m)$ are derived from $(\overleftarrow{\gamma}, [R]^m)$, $(\overrightarrow{\gamma}, [R]^m)$, respectively.

We know by assumption that $[Q]^m \gg_\oplus [Q']^m$. Therefore, $(\overleftarrow{\alpha}, [Q]^m) \gg (\overleftarrow{\alpha}, [Q]^m)$ (which deals with the elements not derived from $[Q]^m$ or $[Q']^m$) and $[Q]^m \gg_{\mathrm{mul}} [Q']^m$. The latter multiset comparison can be established by comparing various elements of $[Q]^m$ and $[Q']^m$ using equality and $\gg$. These comparisons carry over to the derived elements in $[\overleftarrow{\alpha} \cdot Q]^m$ and $[\overleftarrow{\alpha} \cdot Q']^m$: Let $s = (\overleftrightarrow{\gamma}, [R]^m) \in [Q]^m$ and $t = (\overleftrightarrow{\gamma}', [R']^m) \in [Q']^m$. If $s = t$ then the derived elements are also equal. If $s \gg t$, there is only one interesting case: $\gamma = \gamma'$, $s = (\overrightarrow{\gamma}, [R]^m)$ and $t = (\overrightarrow{\gamma}, [R']^m)$, with derived elements $s' = (\overrightarrow{\gamma}, [\overleftarrow{\alpha} \cdot R]^m)$ and $t' = (\overrightarrow{\gamma}, [\overleftarrow{\alpha} \cdot R']^m)$. Since R is a proper subproof of Q, if $[R]^m \succ_\oplus [R']^m$, we conclude that $[\overleftarrow{\alpha} \cdot R]^m \succ_\oplus [\overleftarrow{\alpha} \cdot R]^m$ by the induction hypothesis. This concludes the proof of $[\overleftarrow{\alpha} \cdot Q]^m \gg_{\mathrm{mul}} [\overleftarrow{\alpha} \cdot Q']^m$.

Therefore, $\overleftarrow{\alpha} \cdot Q \succ_\bullet \overleftarrow{\alpha} \cdot Q'$ by definition and $P \cdot Q \succ_\bullet P \cdot Q'$ by the induction hypothesis.

Monotonicity of concatenation in its first argument now follows because $Q \succ_\bullet Q'$ implies $P^{-1} \cdot Q^{-1} \succ_\bullet P^{-1} \cdot (Q')^{-1}$. Inverting both sides yields $Q \cdot P \succ_\bullet Q' \cdot P$. □

3 Decreasing Diagrams

In this section, use the proof string order of Section 2 to give an alternative proof for the conversion version of the decreasing diagram technique [8]. First we establish the corresponding result for proof strings. We let S^* be the Kleene star of S and $S^= = S \cup \{\epsilon\}$.

Lemma 8. *The proof strings corresponding to the peaks and joins of locally decreasing diagrams are decreasing with respect to $\succ_\bullet$, that is,*

1. *if $P = \overleftarrow{\alpha}$ and $Q \in \overleftrightarrow{\gamma\alpha}^*$ then $P \succ_\bullet Q$, and*
2. *if $P = \overleftarrow{\alpha} \cdot \overrightarrow{\gamma}$ and $Q \in (\overleftrightarrow{\gamma\alpha})^* \cdot (\overrightarrow{\gamma})^= \cdot (\overleftrightarrow{\gamma\alpha, \gamma})^* \cdot (\overleftarrow{\alpha})^= \cdot (\overleftrightarrow{\gamma\gamma})^*$ then $P \succ_\bullet Q$.*

Proof. In both cases we show that $[P]^m \succ_\oplus [Q]^m$ by induction on the length of Q.

1. We have $[Q]^m = \{\!\{(\overleftrightarrow{\gamma_1}, t_1), \ldots, (\overleftrightarrow{\gamma_n}, t_n)\}\!\}$ for some n, $\overleftrightarrow{\gamma_i}$ and t_i. By the induction hypothesis, the multisets t_i satisfy $[P]^m \succ_\oplus t_i$, since they correspond to proper subproofs of Q. Because $\overleftarrow{\alpha} \succ \overleftrightarrow{\gamma_i}$ in the precedence for all i, we conclude that $\{\!\{(\overleftarrow{\alpha}, \varnothing)\}\!\} \succ_\oplus \{\!\{(\overleftrightarrow{\gamma_i}, t_i) \mid 1 \le i \le n\}\!\}$, which is equivalent to our claim, $[P]^m \succ_\oplus [Q]^m$.

2. We have
$$[P]^m = \{\!\{(\overleftarrow{\alpha}, \{\!\{(\overrightarrow{\gamma}, \varnothing)\}\!\}), (\overrightarrow{\gamma}, \{\!\{(\overleftarrow{\alpha}, \varnothing)\}\!\})\}\!\},$$
and for some $G \in (\overleftrightarrow{\gamma\gamma})^*$ and $A \in (\overleftrightarrow{\gamma\alpha})^*$,
$$[Q]^m = \{\!\{\ldots, (\overleftarrow{\alpha}, [G]^m), \ldots, (\overrightarrow{\gamma}, [A]^m), \ldots\}\!\}$$
with the $(\overleftarrow{\alpha}, [G]^m)$ or $(\overrightarrow{\gamma}, [A]^m)$ elements possibly missing. For all elements $(\overleftrightarrow{\eta}, u) \in [Q]^m$, $u = [R]^m$ for a proper subproof R of Q, and therefore $[P]^m \succ_\oplus u$ holds by the induction hypothesis. All omitted elements are of the shape $(\overleftrightarrow{\eta}, u)$ with $\alpha \succ \eta$ or $\gamma \succ \eta$, and compare less to one of $(\overleftarrow{\alpha}, \{\!\{\overrightarrow{\gamma}, \varnothing\}\!\})$ or $(\overrightarrow{\gamma}, \{\!\{\overleftarrow{\alpha}, \varnothing\}\!\})$ lexicographically. The remaining (up to two) elements are also dominated by elements of $[P]^m$: By the first part of the proof, $(\overleftarrow{\alpha}, [G]^m)$ is less than $(\overleftarrow{\alpha}, [\overrightarrow{\gamma}]^m)$ and $(\overrightarrow{\gamma}, [A]^m)$ is less than $(\overrightarrow{\gamma}, [\overleftarrow{\alpha}]^m)$. □

Theorem 9 ([8, Theorem 3])**.** *We are given a family of abstract rewrite systems $(\underset{\alpha}{\longrightarrow})_{\alpha \in \mathcal{L}}$. Assume that all local peaks can be joined decreasingly, i.e., for all $\alpha, \gamma \in \mathcal{L}$,*

$$\underset{\alpha}{\longleftarrow} \cdot \underset{\gamma}{\longrightarrow} \subseteq \overset{*}{\underset{\gamma\alpha}{\longleftrightarrow}} \cdot \overset{=}{\underset{\gamma}{\longrightarrow}} \cdot \overset{*}{\underset{\gamma\alpha,\gamma}{\longleftrightarrow}} \cdot \overset{=}{\underset{\alpha}{\longleftarrow}} \cdot \overset{*}{\underset{\gamma\gamma}{\longleftrightarrow}}$$

Then $\underset{\mathcal{L}}{\longrightarrow} = \bigcup_{\alpha \in \mathcal{L}} \underset{\alpha}{\longrightarrow}$ is Church-Rosser.

Proof. We map each rewrite proof to the string obtained by considering just the directions and labels of the proof steps in the proof, mapping $s \underset{\alpha}{\leftarrow} t$ to $\overleftarrow{\alpha}$ and $s \underset{\alpha}{\rightarrow} t$ to $\overrightarrow{\alpha}$. If the rewrite proof has a local peak, then we can replace it by the corresponding joining sequence from a decreasing diagram. The strings P, corresponding to the local peak, and Q, for the joining sequence, satisfy $[P]^m \succ_\bullet [Q]^m$ by Lemma 8. By monotonicity (Lemma 7), this comparison extends to the whole proof string. Because $\succ_\bullet$ is well-founded, this process must terminate in some normal form. This normal form will be a valley proof that is equivalent to the original rewrite proof. □

4 Towards Church-Rosser Modulo

In this section we sketch two approaches to deal with confluence modulo by mapping proofs to proof strings. The first approach is to use the previously developed order directly. The second approach is to extend the proof strings by introducing symmetric proof steps, $\overline{\alpha}$, and incorporate them into the definitions of $\succ_\bullet$. Due to space constraints, we can only sketch it below. There are many notions of confluence modulo (see [7]). We use the following one.

Definition 10. Let $\rightarrow$ and $\vdash\!\dashv$ be abstract rewrite relations, where $\vdash\!\dashv$ is symmetric. We say that $\rightarrow$ is Church-Rosser modulo $\overset{*}{\vdash\!\dashv}$ if

$$(\leftrightarrow \cup \vdash\!\dashv)^* \subseteq \xrightarrow{*} \cdot \overset{*}{\vdash\!\dashv} \cdot \xleftarrow{*}.$$

In the proof transformation setting, this means that whenever we have a subproof $\leftarrow \cdot \rightarrow$, $\vdash\!\dashv \cdot \rightarrow$ or $\leftarrow \cdot \vdash\!\dashv$, we must be able to replace it by a different subproof.

First we sketch how one can use the order from Section 2 directly. Then all proof steps in $\vdash\!\dashv$ must be directed and labeled. In that case, in addition to removing local peaks (except those between $\vdash\!\dashv$ steps, i.e., from $\vdash\!\dashv \cdot \vdash\!\dashv$) one also has to eliminate subproofs of the shape $\underset{\alpha}{\rightarrow} \cdot \underset{\gamma}{\rightarrow}$ whenever $\underset{\alpha}{\rightarrow}$ is a $\vdash\!\dashv$ step and $\underset{\gamma}{\rightarrow}$ is a $\rightarrow$ step. The following lemma shows that any proof in $\xleftarrow[\Upsilon\alpha]{*} \cdot \xrightarrow[\gamma]{=} \cdot \xleftarrow[\Upsilon\alpha,\gamma]{*}$ is a suitable replacement in this case.

Lemma 11. *If* $P = \overrightarrow{\alpha} \cdot \overrightarrow{\gamma}$ *and* $Q \in (\overleftrightarrow{\Upsilon\alpha})^* \cdot (\overrightarrow{\gamma})^= \cdot (\overleftrightarrow{\Upsilon\alpha,\gamma})^*$*, then* $P \succ_\bullet Q$.

Proof. Similar to Lemma 8. □

We have not yet investigated this approach in detail. In this note, Lemma 11 only serves as a point of reference to show that introducing undirected proof steps is useful.

So let us turn to the second approach. We extend the order $\succ_\bullet$ by introducing *undirected* proof steps: Let $\overline{\alpha}$ denote an undirected proof step that is symmetric: $(\overline{\alpha})^{-1} = \overline{\alpha}$. For $[\cdot]^m$, in addition to $[\cdot]^l$ and $[\cdot]^r$, we need an operation $[\cdot]^u$ that collects these undirected proof steps. So we define

- $[\epsilon]^l = \varnothing$, $[\overleftarrow{\alpha} \cdot P]^l = \{\!\{(\overleftarrow{\alpha}, [P]^m)\}\!\} \cup [P]^l$, $[\overrightarrow{\alpha} \cdot P]^l = [P]^l$, $[\overline{\alpha} \cdot P]^l = [P]^l$,
- $[\epsilon]^r = \varnothing$, $[P \cdot \overrightarrow{\alpha}]^r = \{\!\{(\overrightarrow{\alpha}, [P]^m)\}\!\} \cup [P]^r$, $[P \cdot \overleftarrow{\alpha}]^r = [P]^r$, $[P \cdot \overline{\alpha}]^r = [P]^r$,
- $\epsilon^u = \varnothing$, $[\overline{\alpha} \cdot P]^u = \{\!\{(\overline{\alpha}, \varnothing)\}\!\} \cup [P]^u$, $[\overleftarrow{\alpha} \cdot P]^u = [P]^u$, $[\overrightarrow{\alpha} \cdot P]^u = [P]^u$, and
- $[P]^m = [P]^l \cup [P]^r \cup [P]^u$.

Variables $\overleftarrow{\alpha}$ can now equal $\overline{\alpha}$. Correspondingly we define $\overleftrightarrow{S} = \{\overleftarrow{\alpha}, \overrightarrow{\alpha}, \overline{\alpha} \mid \alpha \in S\}$. Even with these changed notions (which result in extended definitions for $\succ_\oplus$ and $\succ_\bullet$), Theorem 7 and Lemmata 4, 8 and 11 remain valid. The following lemma shows how one can eliminate subproofs of the shape $\overline{\alpha} \cdot \overrightarrow{\gamma}$.

Lemma 12. *Let $P = \overline{\alpha} \cdot \overrightarrow{\gamma}$. The following statements are true.*

1. *If $Q \in (\overleftarrow{\Upsilon\alpha \cap \Upsilon\gamma})^* \cdot (\overrightarrow{\gamma})^= \cdot (\overleftarrow{\Upsilon\gamma})^* \cdot \overline{\alpha} \cdot (\overleftarrow{\Upsilon\gamma})^*$ then $P \succ_\bullet Q$.*
2. *If $Q \in (\overleftarrow{\Upsilon\alpha})^* \cdot (\overrightarrow{\gamma})^= \cdot (\overleftarrow{\Upsilon\alpha, \gamma})^*$ then $P \succ_\bullet Q$.*

Proof. Similar to Lemma 8. □

Lemma 12 adds some flexibility over Lemma 11 (where we use a directed α step instead of the undirected one), at the cost of reduced flexibility for eliminating peaks $\overleftarrow{\alpha} \cdot \overrightarrow{\gamma}$ (where again we use a directed α step, but this time pointing left, cf. Lemma 8).

Theorem 13. *Let $\mathcal{L}$ be a set of labels equipped with a well-founded order $\succ$. Furthermore, let $(\underset{\alpha}{\longrightarrow})_{\alpha\in\mathcal{L}}$ and $(\underset{\alpha}{\vdash\!\!\dashv})_{\alpha\in\mathcal{L}}$ be families of abstract rewrite relations, where each $\underset{\alpha}{\vdash\!\!\dashv}$ is symmetric. If*

$$\underset{\alpha}{\longleftarrow} \cdot \underset{\gamma}{\longrightarrow} \subseteq \overset{*}{\underset{\Upsilon\alpha}{\Longleftrightarrow}} \cdot \overset{=}{\underset{\gamma}{\longrightarrow}} \cdot \overset{*}{\underset{\Upsilon\alpha,\gamma}{\Longleftrightarrow}} \cdot \overset{=}{\underset{\alpha}{\longleftarrow}} \cdot \overset{*}{\underset{\Upsilon\gamma}{\Longleftrightarrow}}$$

and

$$\underset{\alpha}{\vdash\!\!\dashv} \cdot \underset{\gamma}{\longrightarrow} \subseteq \left(\overset{*}{\underset{\Upsilon\alpha\cap\Upsilon\gamma}{\Longleftrightarrow}} \cdot \overset{=}{\underset{\gamma}{\longrightarrow}} \cdot \overset{*}{\underset{\Upsilon\gamma}{\Longleftrightarrow}} \cdot \underset{\alpha}{\vdash\!\!\dashv} \cdot \overset{*}{\underset{\Upsilon\gamma}{\Longleftrightarrow}} \right) \cup \left(\overset{*}{\underset{\Upsilon\alpha}{\Longleftrightarrow}} \cdot \overset{=}{\underset{\gamma}{\longrightarrow}} \cdot \overset{*}{\underset{\Upsilon\alpha,\gamma}{\Longleftrightarrow}} \right),$$

for all $\alpha, \gamma \in \mathcal{L}$, where $\underset{\alpha}{\Leftrightarrow} = \underset{\alpha}{\longleftarrow} \cup \underset{\alpha}{\vdash\!\!\dashv} \cup \underset{\alpha}{\longrightarrow}$, then $\underset{\mathcal{L}}{\longrightarrow}$ is Church-Rosser modulo $\overset{*}{\underset{\mathcal{L}}{\vdash\!\!\dashv}}$.

Proof. The proof proceeds in the same way as that of Theorem 9: Whenever a given rewrite proof contains a peak of the shapes $\underset{\alpha}{\longleftarrow} \cdot \underset{\gamma}{\longrightarrow}$, $\underset{\alpha}{\vdash\!\!\dashv} \cdot \underset{\gamma}{\longrightarrow}$, or $\underset{\gamma}{\longleftarrow} \cdot \underset{\alpha}{\vdash\!\!\dashv}$, we can find a replacement proof by assumption. Considering the corresponding proof strings, the replacement is smaller than the peak with respect to $\succ_\bullet$. This extends to the whole strings by monotonicity. By well-foundedness of $\succ_\bullet$, this process will terminate. It is easy to see that the resulting normal forms are of the shape $\overset{*}{\underset{\mathcal{L}}{\longrightarrow}} \cdot \overset{*}{\underset{\mathcal{L}}{\vdash\!\!\dashv}} \cdot \overset{*}{\underset{\mathcal{L}}{\longleftarrow}}$, which establishes Church-Rosser modulo. □

Corollary 14 (Main Theorem of [7])**.** *Let $\mathcal{L}$ be a set of labels equipped with a well-founded order $\succ$. Furthermore, let $(\underset{\alpha}{\longrightarrow})_{\alpha\in\mathcal{L}}$ be a family of abstract rewrite relations and $\vdash\!\!\dashv$ be a symmetric relation. Then $\underset{\mathcal{L}}{\longrightarrow}$ is Church-Rosser modulo $\overset{*}{\vdash\!\!\dashv}$, if for all $\alpha, \gamma \in \mathcal{L}$*

$$\underset{\alpha}{\longleftarrow} \cdot \underset{\gamma}{\longrightarrow} \subseteq \overset{*}{\underset{\Upsilon\alpha}{\longrightarrow}} \cdot \overset{=}{\underset{\gamma}{\longrightarrow}} \cdot \overset{*}{\underset{\Upsilon\alpha,\gamma}{\longrightarrow}} \cdot \overset{*}{\vdash\!\!\dashv} \cdot \overset{*}{\underset{\Upsilon\alpha,\gamma}{\longleftarrow}} \cdot \overset{=}{\underset{\alpha}{\longleftarrow}} \cdot \overset{*}{\underset{\Upsilon\gamma}{\longleftarrow}}$$

and

$$\vdash\!\!\dashv \cdot \underset{\gamma}{\longrightarrow} \subseteq \overset{*}{\underset{\Upsilon\alpha}{\longrightarrow}} \cdot \overset{*}{\vdash\!\!\dashv} \cdot \overset{*}{\underset{\Upsilon\alpha}{\longleftarrow}} \cdot \overset{=}{\underset{\alpha}{\longleftarrow}}.$$

Proof. Apply Theorem 13 using $\mathcal{L}' = \mathcal{L} \cup \{\bot\}$ with $\alpha \succ \bot$ for all $\alpha \in \mathcal{L}$ as labels, and label all $\vdash\!\!\dashv$ steps by $\bot$. □

As another instance of Theorem 13 we can obtain a key lemma for abstract Church-Rosser modulo from [1]:

Corollary 15 ([1, Lemma 2.1])**.** *Let $(\underset{\alpha}{\longrightarrow})_{\alpha\in\mathcal{L}}$ and $(\underset{\alpha}{\vdash\!\!\dashv})_{\alpha\in\mathcal{L}}$ be families of abstract rewrite relations, where each $\underset{\alpha}{\vdash\!\!\dashv}$ is symmetric. Then $\underset{\mathcal{L}}{\longrightarrow}$ is Church-Rosser modulo $\overset{*}{\underset{\mathcal{L}}{\vdash\!\!\dashv}}$, if for all $\alpha, \gamma \in \mathcal{L}$,*

$$\underset{\alpha}{\longleftarrow} \cdot \underset{\gamma}{\longrightarrow} \subseteq \underset{\Upsilon\alpha,\gamma}{\Longleftrightarrow} \quad \text{and} \quad \underset{\alpha}{\vdash\!\!\dashv} \cdot \underset{\gamma}{\longrightarrow} \subseteq \underset{\Upsilon\alpha,\gamma}{\Longleftrightarrow}.$$

5 A Note on Witnesses for Non-Confluence

In this section we present an example of a *linear*, non-confluent TRS whose critical pairs are nevertheless *deeply joinable*. Two terms s, t are deeply joinable if any reducts $s \to^* s'$, $t \to^* t'$ are joinable. The example shows that when looking for witnesses for non-confluence, it does not suffice to look at reducts of critical pairs alone; some smarter technique is required in general.

Example 4. The TRS $\mathcal{R}$ consists of the rules

$$\begin{array}{llll} \mathsf{f}(\mathsf{u}(\mathsf{O}),\mathsf{u}(y)) \to \mathsf{A} & \mathsf{O} \to \mathsf{u}(\mathsf{O}) & \mathsf{u}(x) \to x & \mathsf{f}(x,y) \to \mathsf{f}(x,\mathsf{u}(y)) \\ \mathsf{f}(\mathsf{v}(x),\mathsf{v}(\mathsf{O})) \to \mathsf{B} & \mathsf{O} \to \mathsf{v}(\mathsf{O}) & \mathsf{v}(x) \to x & \mathsf{f}(x,y) \to \mathsf{f}(\mathsf{v}(x),y) \end{array}$$

This TRS is not confluent since $\mathsf{A} \mathrel{{}^*_{\mathcal{R}}\!\!\leftarrow} \mathsf{f}(\mathsf{O},\mathsf{O}) \mathrel{\to^*_{\mathcal{R}}} \mathsf{B}$. There are 12 critical pairs, but they originate from only 4 different sources. We consider each possible source in turn.

1. $\mathsf{f}(\mathsf{u}(\mathsf{O}),\mathsf{u}(y))$ (5 critical pairs). By induction we can show that any term reachable from $\mathsf{f}(\mathsf{u}(\mathsf{O}),\mathsf{u}(y))$ is either equal to A or has shape $\mathsf{f}(\{\mathsf{u},\mathsf{v}\}^*(\mathsf{O}),\mathsf{u}^*(y)\})$, which in turn can be reduced to A. Therefore, all the corresponding critical pairs are deeply joinable.
2. $\mathsf{f}(\mathsf{v}(x),\mathsf{v}(\mathsf{O}))$ (5 critical pairs). This is analogous to the previous case, swapping the arguments of f and the roles of u and v.
3. $\mathsf{f}(x,y)$ (1 critical pair). From this source, we can reach only terms of shape $\mathsf{f}(\mathsf{v}^*(x),\mathsf{u}^*(y))$, which can all be rewritten to $\mathsf{f}(x,y)$.
4. O (1 critical pair). We can rewrite O to $\{\mathsf{u},\mathsf{v}\}^*(\mathsf{O})$, all of which reduce back to O.

Therefore, all critical pairs of $\mathcal{R}$ are deeply joinable as desired.

It has been pointed to the author that $\mathcal{R}$ is E-overlapping [4]. It is an interesting question whether this can be avoided. Note, however, that existence of E-overlaps is undecidable.

Acknowledgments. The author is grateful to the anonymous reviewers for constructive feedback on this paper, to Vincent van Oostrom for sharing and discussing related work on involutive monoids, and to Aart Middeldorp and Harald Zankl for fruitful discussions.

References

[1] T. Aoto and Y. Toyama. A reduction-preserving completion for proving confluence of non-terminating term rewriting systems. *LMCS*, 8(1:31):1–29, 2012.

[2] L. Bachmair and N. Dershowitz. Equational inference, cannonical proofs, and proof orderings. *JACM*, 41(2):236–276, 1994.

[3] M.C.F. Ferreira. *Termination of Term Rewriting: Well-Foundedness, Totality and Transformations.* PhD thesis, Utrecht University, 1995.

[4] H. Gomi, M. Oyamaguchi, and Y. Ohta. On the Church-Rosser property of root-E-overlapping and strongly depth-preserving term rewriting systems. *Trans. IPSJ*, 39(4):992–1005, 1998.

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

[6] J.-P. Jouannaud and V. van Oostrom. Diagrammatic confluence and completion. In *Proc. 36th ICALP*, volume 5556 of *LNCS*, pages 212–222, 2009.

[7] E. Ohlebusch. Church-Rosser theorems for abstract reduction modulo an equivalence relation. In *Proc. 9th RTA*, volume 1379 of *LNCS*, pages 17–31, 1998.

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

Confluence of Non-Left-Linear TRSs via Relative Termination (Extended Abstract)*

Dominik Klein
JAIST, Japan
dominik.klein@jaist.ac.jp

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

Abstract

We present a confluence criterion for term rewrite systems by relaxing termination requirements of Knuth and Bendix' confluence criterion, using joinability of extended critical pairs. Because computation of extended critical pairs requires equational unification, which is undecidable, we give a sufficient condition for testing joinability automatically.

1 Introduction

Knuth and Bendix [8] showed that confluence of terminating TRSs is decidable by testing joinability of critical pairs, which are induced by *overlaps*. In the case of non-termination, several powerful techniques have been developed for proving confluence of left-linear systems (cf. [1]). Still, proving confluence of TRSs that are both non-left-linear and non-terminating remains challenging.

Results that tackle this setting can be roughly classified into three categories: First, by generalizing the notion of overlaps, one can formulate *direct criteria* that guarantee confluence [4]. The second approach is to *decompose* a TRS into smaller ones, show confluence of each of them by existing criteria, and formulate modularity conditions to ensure that the union remains confluent (cf. [1]). The third approach is to generalize Knuth and Bendix' confluence criterion by relaxing termination requirements to *relative termination*. A famous result here is Jouannaud and Kirchner's criterion for the Church-Rosser modulo property [6] based on extended critical pairs. Geser [3] analyzed their proof to derive confluence criteria based only on syntactical critical pairs.

We present a new confluence criterion that also relies on relative termination, and can be applied to non-left-linear TRSs. The criterion requires to check joinability of extended critical pairs, but we show that under certain conditions, joinability can be concluded from joinability of syntactical critical pairs. With it, we are able to fully automatically prove confluence of several non-terminating, non-left-linear TRSs, for which no known criteria exist.

2 Preliminaries

We assume familiarity with the basics of term rewriting ([2]).

Term Rewriting. Terms are inductively defined over a set $\mathcal{F}$ of fixed-arity *function symbols*, and a set $\mathcal{V}$ of *variables*. A rewrite rule $\ell \to r$ is a pair (ℓ, r) of terms with $\mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(\ell)$ and $\ell \notin \mathcal{V}$. A TRS is a set of rewrite rules. An *extended rewrite rule* is a pair (ℓ, r) of terms with $\ell \notin \mathcal{V}$, and an *extended TRS* (*eTRS*) is a set of extended rewrite rules. We write $\downarrow_{\mathcal{R}}$ for the *join* relation $\to_{\mathcal{R}}^* \cdot {}_{\mathcal{R}}^*\!\leftarrow$. We write $\to_{\mathcal{R}/\mathcal{S}}$ for $\to_{\mathcal{S}}^* \cdot \to_{\mathcal{R}} \cdot \to_{\mathcal{S}}^*$. A TRS $\mathcal{R}$ is *relatively terminating* over a TRS $\mathcal{S}$ or $\mathcal{R}/\mathcal{S}$ is terminating, if $\to_{\mathcal{R}/\mathcal{S}}$ is so.

*This work is supported by the Grant-in-Aids for Young Scientists (B) 22700009 and Scientific Research (B) 23300005 of the Japan Society for the Promotion of Science. The full version of this abstract appeared as [7].

Unification. Let $\mathcal{S}$ be a set of equalities (or a TRS) and s, t terms over variables X. An *$\mathcal{S}$-unifier* of s and t is a substitution σ such that $s\sigma \leftrightarrow^*_{\mathcal{S}} t\sigma$. A substitution σ is *$\mathcal{S}$-more general* than a substitution σ' on X ($\sigma \lesssim^X_{\mathcal{S}} \sigma'$), if there exists a substitution τ such that $x\sigma' \leftrightarrow^*_{\mathcal{S}} x\sigma\tau$ for all $x \in X$. Let σ be an $\mathcal{S}$-unifier of s and t. The substitution σ is an *$\mathcal{S}$-most general unifier* (*$\mathcal{S}$-mgu*) of s and t, if σ is $\mathcal{S}$-more general than any $\mathcal{S}$-unifier of s and t on X. In the special case of $\mathcal{S} = \varnothing$, we simply speak of (syntactic) unification, unifiers and mgu's. Note that in general $\mathcal{S}$-unifiability does not ensure the existence of an $\mathcal{S}$-mgu.

Critical Pairs. Let $\mathcal{R}_1, \mathcal{R}_2, \mathcal{S}$ be eTRSs. An $\mathcal{S}$-overlap $(\ell_1 \to r_1, p, \ell_2 \to r_2)_\sigma$ of $\mathcal{R}_1$ on $\mathcal{R}_2$ consists of a variant $\ell_1 \to r_1$ of a rule in $\mathcal{R}_1$ and a variant $\ell_2 \to r_2$ of a rule in $\mathcal{R}_2$, a non-variable position p in ℓ_2 and a substitution σ, such that $\ell_1\sigma \leftrightarrow^*_{\mathcal{S}} \ell_2|_p\sigma$. If p is the root position ε, then $\ell_1 \to r_1$ and $\ell_2 \to r_2$ may not be variants of each other. The pair $(\ell_2\sigma[r_1\sigma]_p, r_2\sigma)$ induced from the overlap is an *$\mathcal{S}$-extended critical pair* (or simply *$\mathcal{S}$-critical pair*) of $\ell_1 \to r_1$ and $\ell_2 \to r_2$ at p, written $\ell_2\sigma[r_1\sigma]_p \ {}_{\mathcal{R}_1}\!\leftarrow_{\mathcal{S}} \!\mathord{\times}\!\rightarrow_{\mathcal{R}_2} r_2\sigma$. We call $\varnothing$-overlaps *syntactic overlaps* (or just *overlaps*). Remark that our definition of $\mathcal{S}$-critical pairs includes pairs originating from non-minimal unifiers, which are usually excluded from the definition to guarantee finiteness of critical pairs. Knuth and Bendix' criterion [8] states that a terminating TRS $\mathcal{R}$ is confluent if ${}_{\mathcal{R}}\!\leftarrow_{\varnothing} \!\mathord{\times}\!\rightarrow_{\mathcal{R}} \subseteq \downarrow_{\mathcal{R}}$.

Let $\mathsf{REN}(t)$ denote a linear term resulting from replacing in t each variable occurrence by a fresh variable. We write $\widehat{\mathcal{R}}$ for the eTRS $\{\mathsf{REN}(\ell) \to r \mid \ell \to r \in \mathcal{R}\}$. Two TRSs $\mathcal{R}$ and $\mathcal{S}$ are *strongly non-overlapping*, denoted $\mathsf{SNO}(\mathcal{R}, \mathcal{S})$, if $\widehat{\mathcal{S}}$ has no syntactic overlaps on $\widehat{\mathcal{R}}$ and $\widehat{\mathcal{R}}$ has no syntactic overlaps on $\widehat{\mathcal{S}}$. Consider, for instance, $\mathcal{R} = \{\mathsf{f}(x, x) \to \mathsf{a}\}$ and $\mathcal{S} = \{\mathsf{f}(\mathsf{g}(x), x) \to \mathsf{g}(x)\}$. We have $\widehat{\mathcal{R}} = \{\mathsf{f}(y, z) \to \mathsf{a}\}$ and $\widehat{\mathcal{S}} = \{\mathsf{f}(\mathsf{s}(y), z) \to \mathsf{g}(x)\}$. Because there is an overlap of $\widehat{\mathcal{R}}$ on $\widehat{\mathcal{S}}$ at the root position, $\mathsf{SNO}(\mathcal{R}, \mathcal{S})$ does not hold. Note that there are no overlaps of $\mathcal{R}$ on $\mathcal{S}$.

Decreasing Diagrams. We recall the decreasing diagram technique [10] which is a powerful confluence criterion for abstract rewrite systems (ARSs). Let $\mathcal{A} = (A, \langle \to_\alpha \rangle_{\alpha \in I})$ be an ARS and $>$ a proper order on I. For every $\alpha \in I$ we write $\xrightarrow[\vee]{}_\alpha$ for $\{\to_\beta \mid \beta \in I \text{ and } \beta < \alpha\}$, and $\xrightarrow[\vee]{*}_\alpha$ for $(\xrightarrow[\vee]{}_\alpha)^*$. The union of $\xrightarrow[\vee]{}_\alpha$ and ${}_\alpha\!\xleftarrow[\vee]{}$ is denoted by $\xleftrightarrow[\vee]{}_\alpha$. For $\alpha, \beta \in I$, the union of $\xrightarrow[\vee]{}_\alpha$ and $\xrightarrow[\vee]{}_\beta$ is written as $\xrightarrow[\vee]{}_{\alpha\beta}$. Two labels α and β are *decreasing* with respect to $>$ if

$$ {}_\alpha\!\leftarrow \cdot \rightarrow_\beta \ \subseteq \ \xleftrightarrow[\vee]{*}_\alpha \cdot \rightarrow^{=}_\beta \cdot \xleftrightarrow[\vee]{*}_{\alpha\beta} \cdot \ {}^{=}_\alpha\!\leftarrow \cdot \ {}^{*}_\beta\!\xleftarrow[\vee]{} $$

An ARS $\mathcal{A} = (A, \langle \to_\alpha \rangle_{\alpha \in I})$ is *decreasing* if there exists a well founded order $>$ such that every two labels in I are decreasing with respect to $>$.

Theorem 2.1 ([10])**.** *Every decreasing ARS is confluent.* □

3 Confluence Criterion

We present a generalisation of Knuth and Bendix' confluence criterion. Below, $\mathcal{R}$ and $\mathcal{S}$ stand for TRSs.

Theorem 3.1. *Suppose that $\mathcal{S}$ is confluent, $\mathcal{R}/\mathcal{S}$ is terminating, and $\mathsf{SNO}(\mathcal{R}, \mathcal{S})$. The union $\mathcal{R} \cup \mathcal{S}$ of the TRSs is confluent if and only if ${}_{\mathcal{R}}\!\leftarrow_{\mathcal{S}} \!\mathord{\times}\!\rightarrow_{\mathcal{R}} \subseteq \downarrow_{\mathcal{R}\cup\mathcal{S}}$.*

Note that Theorem 3.1 is equivalent to Knuth and Bendix' criterion when taking $\mathcal{S} = \varnothing$. In order to prove the theorem we introduce an *intermediate* relation $\twoheadrightarrow$, such that $\to_{\mathcal{R}\cup\mathcal{S}} \subseteq$

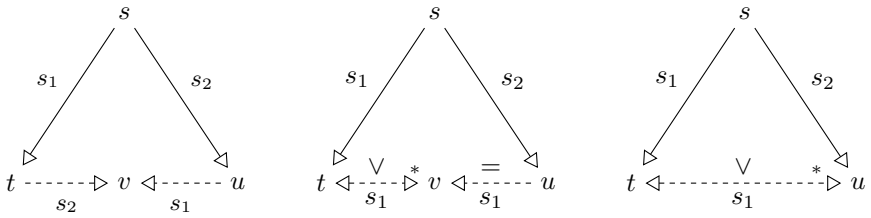


Figure 1: Decreasingness of $\rightarrowtriangle$.

$\rightarrowtriangle \subseteq \rightarrow^*_{\mathcal{R}\cup\mathcal{S}}$. Confluence of this intermediate relation readily implies confluence of $\mathcal{R}\cup\mathcal{S}$. The relation $\rightarrowtriangle$ is defined as the union of $\rightarrow_{\mathcal{R}_\mathcal{S}}$ and $\rightarrow^*_{\mathcal{S}}$, where $\mathcal{R}_\mathcal{S}$ is the TRS

$$\{\ell'\sigma \to r\tau \mid \ell'\rho \to r \in \mathcal{R} \text{ and } \sigma \rightarrow^*_{\mathcal{S}} \rho\tau \text{ for some substitution } \rho \text{ on } \mathcal{V}\ \}$$

Here $\sigma \rightarrow^*_{\mathcal{S}} \tau$ means that $x\sigma \rightarrow^*_{\mathcal{S}} x\tau$ for all variables x. It is important to note that in the definition linearity of ℓ' can be assumed without loss of generality, and that the inclusions $\rightarrow_{\mathcal{R}} \subseteq \rightarrow_{\mathcal{R}_\mathcal{S}} \subseteq \rightarrow^*_{\mathcal{S}} \cdot \rightarrow_{\mathcal{R}}$ hold.

Proof of Theorem 3.1. Since the "only if"-direction is trivial, we only show the "if"-direction. Assume $_{\mathcal{R}}\!\leftarrow_{\mathcal{S}}\!\!\times\!\!\rightarrow_{\mathcal{R}} \subseteq \downarrow_{\mathcal{R}\cup\mathcal{S}}$. Because confluence of $\rightarrowtriangle$ implies confluence of $\mathcal{R}\cup\mathcal{S}$, according to Theorem 2.1 it is enough to show decreasingness of $\rightarrowtriangle$. Here we adopt the predecessor labeling [10]: We write $b \rightarrowtriangle_a c$ if $a \rightarrowtriangle^* b \rightarrowtriangle c$. Labels are compared with respect to $\rightarrow^+_{\mathcal{R}/\mathcal{S}}$, denoted by $>$. Since termination of $\mathcal{R}/\mathcal{S}$ is presupposed in the theorem, the relation $>$ forms a well-founded order. Let $t\ _{s_1}\!\!\leftarrowtriangle s \rightarrowtriangle_{s_2} u$. We distinguish three cases. (a) If $t\ \overset{*}{\mathcal{S}}\!\leftarrow s \rightarrow^*_{\mathcal{S}} u$ then $t \rightarrow^*_{\mathcal{S}} v\ \overset{*}{\mathcal{S}}\!\leftarrow u$ for some v. (b) If $t\ _{\mathcal{R}_\mathcal{S}}\!\leftarrow s \rightarrow^*_{\mathcal{S}} u$ then $t \overset{\vee}{\longleftrightarrow}^*_s v\ \overset{=}{\mathcal{R}_\mathcal{S}}\!\leftarrow u$ for some v. (c) If $t\ _{\mathcal{R}_\mathcal{S}}\!\leftarrow s \rightarrow_{\mathcal{R}_\mathcal{S}} u$ then $t \overset{\vee}{\longleftrightarrow}^*_s u$ for some v. In all cases decreasingness is established, as seen in Figure 1. □

The next examples illustrate Theorem 3.1. Note that no existing confluence tool can prove confluence of the involved TRSs automatically.

Example 3.2. Consider the TRS

$$1\colon\ \mathsf{f}(x,x) \to (x+x)+x \qquad\qquad 2\colon\ x+y \to y+x$$

Take $\mathcal{R} = \{1\}$ and $\mathcal{S} = \{2\}$. One can easily verify $\mathsf{SNO}(\mathcal{R},\mathcal{S})$. Termination of $\mathcal{R}/\mathcal{S}$ can be established using a termination tool such as TTT2 v1.06 [9],[1] and confluence of $\mathcal{S}$ follows from orthogonality. Because of $_{\mathcal{R}}\!\leftarrow_{\mathcal{S}}\!\!\times\!\!\rightarrow_{\mathcal{R}} = \varnothing \subseteq \downarrow_{\mathcal{R}\cup\mathcal{S}}$, we conclude confluence by Theorem 3.1.

Example 3.3. Consider the TRS

$$1\colon\ \mathsf{f}(x,x) \to \mathsf{s}(\mathsf{s}(x)) \qquad\qquad 2\colon\ \infty \to \mathsf{s}(\infty)$$

Take $\mathcal{R} = \{1\}$ and $\mathcal{S} = \{2\}$. As in Example 3.2, one can easily verify the conditions of Theorem 3.1, including $_{\mathcal{R}}\!\leftarrow_{\mathcal{S}}\!\!\times\!\!\rightarrow_{\mathcal{R}} = \varnothing \subseteq \downarrow_{\mathcal{R}\cup\mathcal{S}}$. Hence the TRS is confluent.

Example 3.4. Consider the TRS

$$\begin{array}{lll} 1\colon & \mathsf{eq}(\mathsf{s}(n), x:xs, x:ys) \to \mathsf{eq}(n, xs, ys) & 3\colon\ \ \mathsf{nats} \to 0 : \mathsf{inc}(\mathsf{nats}) \\ 2\colon & \mathsf{eq}(n, xs, xs) \to \mathsf{T} & 4\colon\ \mathsf{inc}(x:xs) \to \mathsf{s}(x) : \mathsf{inc}(xs) \end{array}$$

[1] `http://colo6-c703.uibk.ac.at/ttt2/`

Take $\mathcal{R} = \{1, 2\}$ and $\mathcal{S} = \{3, 4\}$. Again, $\mathsf{SNO}(\mathcal{R}, \mathcal{S})$, termination of $\mathcal{R}/\mathcal{S}$ and confluence of $\mathcal{S}$ are established. Moreover, one can show

$$\mathcal{R}{\leftarrow_{\mathcal{S}}}{\rtimes}{\rightarrow_{\mathcal{R}}} = \{(\mathsf{eq}(s, t, u), \mathsf{T}) \mid s,\ t,\ u \text{ are terms and } t \leftrightarrow_{\mathcal{S}}^* u\}$$

and thus the set is included in $\downarrow_{\mathcal{R}\cup\mathcal{S}}$. Hence by using Theorem 3.1 we conclude that $\mathcal{R} \cup \mathcal{S}$ is confluent.

4 Joinability of $\mathcal{S}$-Critical Pairs

The biggest challenge in applying Theorem 3.1 is to check $\mathcal{R}{\leftarrow_{\mathcal{S}}}{\rtimes}{\rightarrow_{\mathcal{R}}} \subseteq \downarrow_{\mathcal{R}\cup\mathcal{S}}$ automatically. The standard approach is to compute a *minimal complete set* (see [2]) of $\mathcal{S}$-unifiers for ℓ_1 and $\ell_2|_p$ for each combination of rules $\ell_1 \to r_1$, $\ell_2 \to r_2$ and a non-variable position p in ℓ_2. Then, joinability of its induced critical pairs ensures joinability for all $\mathcal{S}$-unifiers. However, depending on $\mathcal{S}$, the computation of minimal complete sets varies, and worse, minimal complete sets may not even exist for $\mathcal{S}$-unifiable terms.

We present a sufficient condition for the joinability of $\mathcal{S}$-critical pairs without performing specific equational unification algorithms. We call a term t *strongly $\mathcal{S}$-stable* if for every non-variable position p in t there are no term u and substitution σ such that $t|_p\sigma \to_{\mathcal{S}}^* \cdot \xrightarrow{\varepsilon}_{\mathcal{S}} u$.

Theorem 4.1. *Let $\mathcal{S}$ be a confluent TRS. An mgu of strongly $\mathcal{S}$-stable terms s and t is an $\mathcal{S}$-mgu of s and t.* □

When automating Theorem 3.1, confluence of $\mathcal{S}$ and $\mathsf{SNO}(\mathcal{R}, \mathcal{S})$ can be assumed. The latter condition implies strong $\mathcal{S}$-stability of ℓ for all $\ell \to r \in \mathcal{R}$. Therefore, according to Theorem 4.1, a syntactical overlap by an mgu μ is also an $\mathcal{S}$-overlap by $\mathcal{S}$-mgu μ. Thus joinability of its syntactical critical pairs implies joinability of $\mathcal{S}$-critical pairs induced by any $\mathcal{S}$-unifier.

Example 4.2 (continued from Example 3.4)**.** We consider again the example with $\mathcal{R} = \{1, 2\}$ and $\mathcal{S} = \{3, 4\}$. Take the first and second rules renamed:

$$1\colon\ \mathsf{eq}(\mathsf{s}(n), x : xs, x : ys) \to \mathsf{eq}(n, xs, ys) \qquad 2\colon\ \mathsf{eq}(m, zs, zs) \to \mathsf{T}$$

We know that there is an overlap between 1 and 2 at root position with the mgu $\mu = \{m \mapsto \mathsf{s}(n),\ zs \mapsto x : xs,\ ys \mapsto xs\}$. Elsewhere, even $\mathcal{S}$-overlaps cannot occur. The induced critical pair $(\mathsf{eq}(n, xs, xs), \mathsf{T})$ is trivially joinable by the second rule. Hence $\mathcal{R}{\leftarrow_{\mathcal{S}}}{\rtimes}{\rightarrow_{\mathcal{R}}} \subseteq \downarrow_{\mathcal{R}\cup\mathcal{S}}$ holds.

5 Experiments

We implemented Theorem 3.1 together with Theorem 4.1. We employed T$_\text{T}$T$_2$ v1.06 [9] for checking relative termination $\mathcal{R}/\mathcal{S}$. To check confluence of $\mathcal{S}$, we used the existing three state-of-the-art confluence provers: ACP v0.20 [1],[2] CSI v0.1 [12],[3] and Saigawa v1.2 [5].[4] Since termination of $\mathcal{R} \cup \mathcal{S}$ cannot be assumed, we only test joinability of $\mathcal{S}$-critical pairs by at most four rewrite steps from each side. By enumeration we searched for a suitable combination of $\mathcal{R}$ and $\mathcal{S}$. Our testbed consists of 32 non-left-linear non-terminating TRSs, which are taken from the Confluence Problem Database (Cops Nos. 1–116)[5] and Examples 3.2, 3.3 and 3.4. The tests were single-threaded run on a system equipped with an Intel Core Duo L7500 with 1.6 GHz and 2 GB of RAM using a timeout of 60 seconds.

[2] http://www.nue.riec.tohoku.ac.jp/tools/acp/
[3] http://cl-informatik.uibk.ac.at/software/csi/
[4] http://www.jaist.ac.jp/project/saigawa/
[5] http://coco.nue.riec.tohoku.ac.jp/

Table 1: Summary of experimental results (32 TRSs)

	ACP	ACP*	CSI	CSI*	Saigawa	Saigawa*
confluence proved	12	**19**	7	**15**	0	**10**
timeout (60 sec)	0	0	5	5	0	0

The results are depicted in Table 1.[6] Here columns ACP, CSI and Saigawa show results for running the respective tools, and ACP*, CSI* and Saigawa* show results when using the respective tool to show confluence of the $\mathcal{S}$-part in Theorem 3.1. It should be noted, that the criteria implemented by Saigawa apply only to left-linear systems, whereas CSI is able to show confluence of non-left-linear systems by order-sorted decomposition [12], and the implementation of ACP includes criteria based on layer preserving and persistency decompositions, and the direct criterion by Gomi et al. [1, 4].

We conclude by stating future work. We used relative termination to handle non-terminating TRSs. However, relative termination still poses a strict restriction. We anticipate that the use of *critical pair steps* [5] relaxes this restriction. Weakening the strong non-overlappingness condition is another future work. Non-root-E-overlappingness [4] and persistence [11] would be candidates for this.

Acknowledgements. We thank the anonymous referees for their valuable comments.

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] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[3] A. Geser. *Relative Termination.* PhD thesis, Universität Passau, 1990. Also available as technical report 91-03 of Ulmer Informatik-Berichte. Universität Ulm.

[4] H. Gomi, M. Oyamaguchi, and Y. Ohta. On the Church-Rosser property of root-E-overlapping and strongly depth-preserving term rewriting systems. *Trans. IPSJ*, 39(4):992–1005, 1998.

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

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

[7] D. Klein and N. Hirokawa. Confluence of non-left-linear TRSs via relative termination. In *Proc. 18th LPAR*, volume 7180 of *LNCS*, pages 258–273, 2012.

[8] D.E. Knuth and P. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. 1970.

[9] M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean Termination Tool 2. In *Proc. 20th RTA*, volume 5595 of *LNCS*, pages 295–304, 2009.

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

[11] R.M. Verma. Unique normal forms and confluence of rewrite systems: Persistence. In *Proc. 14th IJCAI*, volume 1, pages 362–370, 1995.

[12] H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – a confluence tool. In *Proc. 23th CADE*, volume 6803 of *LNAI*, pages 499–505, 2011.

[6] Detailed results are available at `http://www.jaist.ac.jp/project/saigawa/`

IaCOP
Interface for the Administration of Cops*

Christian Nemeth
University of Innsbruck, Austria

Harald Zankl
University of Innsbruck, Austria

Nao Hirokawa
JAIST, Japan

Abstract

IaCOP maintains the problems considered for the competition of confluence tools. It allows the submission of new problems and supports automatic tagging of problems with certain properties. Problems can be filtered by boolean combinations of tags and the web interface comes with a mechanism to select problems for competitions.

1 Introduction

Beginning in the late 1980's the increasing power of computer hardware gave a tremendous boost to the area of automated deduction. Since then many tools have been developed to automatically reason about properties of programs. Nowadays many of these tools are successfully applied in software and hardware verification and are inevitable for the development of complex systems.

To spur the development of these tools and to give fair comparisons between them dedicated competitions take place regularly where tools compete against each other on a fixed (but previously typically unknown) selection of problems. Dating back to 1996 CASC[1] [4] has the longest tradition among such competitions. In CASC several provers try to prove or disprove validity of first-order logic formulas. Albeit undecidable in general, modern tools can establish a result in surprisingly many cases automatically. The first edition of the SAT competition[2] took place in 2002 where various SAT solvers investigate satisfiability of propositional logic formulas. Starting with a demo in 2003 the termination competition[3] emerged where tools try to prove (refute) termination of programs automatically. While in the early years programs have been solely given as *term rewrite systems* nowadays termination of Prolog and Haskell programs as well as Java Bytecode is considered [2]. Satisfiability modulo Theories (SMT) extends propositional logic with various theories (arrays, bit-vectors, linear arithmetic, ...) and in 2005 the first SMT-COMP was conducted [1].

Obviously there are too many competitions to mention all of them but they have some properties in common. To make a competition thrilling a (large) database of interesting problems is essential and to run a competition within e.g. one day a suitable selection of problems must be performed. On the other hand tool authors want to test their tools on a subset of problems satisfying some given properties.

In 2012 the first edition of CoCo[4] (*Co*nfluence *Co*mpetition) will be run during the 1st International Workshop on Confluence, collocated with the 23rd International Conference on Rewriting Techniques and Applications (RTA 2012). The input problems considered for this competition—Cops,[5] (*Co*nfluence *P*roblem*s*)—will be maintained by IaCOP, a web *I*nterface

*Supported by the Austrian Science Fund (FWF) P22467 and the Grant-in-Aid for Young Scientists (B) 22700009 of the Japan Society for the Promotion of Science.

[1] http://www.cs.miami.edu/~tptp/CASC/
[2] http://www.satcompetition.org/
[3] http://termcomp.uibk.ac.at
[4] http://coco.nue.riec.tohoku.ac.jp/
[5] http://coco.nue.riec.tohoku.ac.jp/cops/

46.trs

```
(VAR x)
(RULES
  F(x,x) -> A
  F(x,G(x)) -> B
  C -> G(C)
)
(COMMENT from p.813 of \cite{Hue80})
```

tags: non_confluent non_left_linear non_linear locally_confluent non_orthogonal non_terminating non_ground

Figure 1: A confluence problem displayed in IaCOP.

for the *A*dministration of *Cop*s [3]. This paper describes the main features of IaCOP, a test version of which is available from

http://termcomp-devel.uibk.ac.at/csam1779/12iwc

Cops is merely an archive of confluence problems (`*.trs`) in the old TRS syntax,[6] and even lacks directory structures. However, IaCOP allows the submission of new problems and supports automatic *tagging* of problems with certain properties (see Figure 1). Problems can be filtered by boolean combinations of tags and the web interface comes with a mechanism to select problems for competitions. Hence IaCOP allows to maintain the database used for CoCo but is independent from the actual software running the competition. We anticipate that this design is more flexible and user friendly than combining both functionalities.

2 Features

IaCOP distinguishes between a plain *user mode* and an *administrator mode*. The functionality for users comprises the submission of new problems, searching of problems and downloading them. The administrator can (in addition) accept newly submitted problems, control the tag setting, and can initiate a selection of problems for competitions (depending on random seeds). Each item of the functionality is described in more detail below.

2.1 User Functionality

Submission: Users can upload single TRSs or zip archives in the old TRS syntax. The correct syntax is evaluated upon upload and only well-formed TRSs are uploaded. Useful auxiliary information (such as pointers to the literature or the origin of a problem) can be submitted in the `COMMENTS` field.

Search: IaCOP allows to search the content of problems as well as their tags by a full text search, i.e., words must be matched exactly. Search terms can be combined by boolean combinations, i.e., conjunctions, disjunctions, and complement. Hierarchic grouping of expressions is possible by braces.

[6] http://www.lri.fr/~marche/tpdb/format.html

Example 1. The search string "`terminating locally_confluent`" yields all problems with tags terminating and locally confluent. To search problems that are neither left-linear nor (known to be) terminating the search string "`!left_linear !terminating`" is used. This can also be expressed by "`!{left_linear OR terminating}`" where braces are used to group expressions.

Since some tags are undecidable (e.g. terminating, confluent), the lack of such a tag may indicate that the problem does not have the property or that it is unknown if the problem has this property. Hence problems are also tagged with the inverse (e.g. non_terminating, non_confluent) if they are known not to satisfy the property. Consequently the search string "`!confluent !non_confluent`" yields all problems whose confluence status is unknown.

Download: The result set of any search can be downloaded either including the tag information in the problem or without it. The former allows to process problems also outside of IaCOP and the latter option is useful to provide test sets for competitions, where tools should not make use of additional tag information.

2.2 Administrator Functionality

Tagging: To assist the administrator, IaCOP can employ external tools to compute the tags of problems automatically. Those tools are started upon upload of a problem. Tags can also be maintained manually by the administrator. Tags can be renamed and new tags can be added to problems at any time.

Problem Management: Once new problems are submitted, the administrator gets notified by email and can then decide to accept these problems into Cops. Afterwards they get attached a unique id and become visible to standard users. While (for backwards compatibility) the actual content of a problem cannot be changed—once it is accepted—this restriction does not apply to tags. However, tags used for competitions (see below) cannot be changed.

Problem Selection: To select a subset of the Cops for competitions the administrator can add a new manual tag in IaCOP. After providing a lower and an upper bound on the number of selected problems and a list of random seeds (which influence the selection) IaCOP tags problems for a competition. For fairness of the whole process the random seeds should be contributed by the participants of the competition. For transparency the seeds and the result of the selection are made public immediately and tags used for competitions cannot be changed once attached to a problem.

3 Implementation

To couple the problems as loosely as possible to IaCOP no database is employed. Problem files are stored in the file system of the server running IaCOP as plain text files. Tag information is stored in the comment field of TRS problems. This allows the extraction of problems with all existing tag information from IaCOP, maybe add new tags outside of IaCOP and import the updated problems again.

Since problems are stored in plain text the search functionality is implemented by the command line tool `grep`.

4 Related Work

In this section we report on how other communities handle issues concerning the problem database used for competitions. Our insight varies tremendously depending on the specific competition and is reflected in the discussion below. Hence we only report on TermComp and SMT-COMP for which we have submitted problems in recent years.

TermComp: Problems are submitted via Email and the steering committee decides which problems are added to the database. We are not aware that any submission has ever been rejected. However, submitted problems are not immediately available to the outside world. Every now and then the current problems can be downloaded from the competition website.

SMT-COMP: Problems are submitted via Email and easy problems (which every tool could solve) are ignored. Again, problems are not immediately available to others. However, typically the benchmarks can be downloaded before the competition takes place.

For both competitions we are not aware of sophisticated search functionalities which allow the user to extract a set of problems with specific properties.

Finally we mention the ongoing efforts of Stump, Sutcliff, and Tinelli to build StarExec,[7] a uniform platform for running and managing different competitions. The first edition of CoCo will not be run on StarExec but we hope to use this great service already for its next edition.

5 Conclusion

This note presented IaCOP, an Interface for the Administration of Cops. IaCOP makes the submission of new problems attractive to researchers via an online form and supports the administrator in maintaining the database by automatic support for determining the values of tags. Tags (as well as problem content) can be searched by a full text search, which allows to filter problems with certain properties. Finally, IaCOP can select problems for a competition based on a list of random seeds as input.

Concerning future work, one extension could be to run all current confluence provers on newly submitted problems and show the result immediately to the one uploading these problems.

Acknowledgments We are grateful to Aart Middeldorp and Aaron Stump for their valuable suggestions. We thank the anonymous reviewers for their detailed and helpful comments.

References

[1] C. Barrett, M. Deters, L. de Moura, A. Oliveras, and A. Stump. 6 years of SMT-COMP. *Journal of Automated Reasoning*. doi: 10.1007/s10817-012-9246-5. Online First, to appear.

[2] J. W. (ed.). Report on the termination competition 2008. In *Proc. 10th International Workshop on Termination*, pages 100–111, 2009.

[3] C. Nemeth. Interface for the administration of confluence problems (IaCOP). Bachelor thesis, University of Innsbruck, 2012.

[4] G. Sutcliffe. The CADE-23 automated theorem proving system competition – CASC-23. *AI Communications*, 25(1):49–63, 2012.

[7] http://www.starexec.org/

A Case for Completion Modulo Equivalence

Kristoffer H. Rose

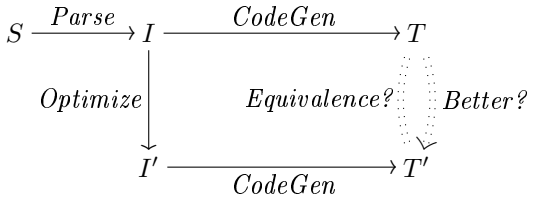
IBM Thomas J. Watson Research Center
P.O.Box 704, Yorktown Heights, NY 10598, USA
krisrose@us.ibm.com

Abstract

We present simple use cases from compiler writing for the use of completion modulo an equational theory, corresponding to the semantics of the target language.

1 Introduction

When working with rewrite systems that specify (and implement) compilers from a program S in a source language through programs I in an intermediate language to programs T in a target language, we are interested in divergent reductions like the following:



(1)

Essentially, in compilers we often permit optimizations that optimize the intermediate language programs I to I' in such a way that when we code generate then we get programs in the target language T and T', which we hope to be semantically equivalent but also that the optimized version is "better," as shown by the two dotted relations at the right of (1), by some metric such as time or space complexity or even code size.

The CRSX system [5] is an implementation of higher order rewriting designed and used specifically to implement compilers [4].[1] One specific extension of CRSX is to use *completion* to implement optimizations, however, the optimizations that are injected into an unoptimizing compiler typically do not preserve the rewrite relation, rather the optimized target programs are expected to be both equivalent and better in the sense of (1) above.

In this extended abstract we give some simple examples of the kind of completion that we have in mind, and hope to open a discussion of what the right framework for proceeding should be!

2 Optimization by Completion

We consider some examples of translation systems with optimizations that introduce critical pairs that are in principle not solvable yet where a completion procedure can be used to incorporate the optimization into a deterministic compiler.

[1] Formally, CRSX is a generalization of the algebraic data types from term rewrite systems and the binding and substitution mechanism of λ calculus, based on the *contraction schemes* of Aczel [1], developed as *combinatory reduction systems* by Klop [2], solidified with van Oostrom and van Raamsdonk [3].

2.1 Notation. All rewriting examples are given in CRSX [5] notation, where terms are built from disjoint sets of *variables* $v, x, \ldots \in \mathcal{V}$, *meta-variables* $\mathrm{M}, \ldots \in \mathcal{M}$, and *constructors* $F, C, \ldots \in \mathcal{C}$ including *literals* $\ell \in \mathcal{L}$, using the syntax

$$
\begin{array}{lr}
t ::= p\ s & \text{(term)} \\
p ::= \ \mid \{\ k : t; \ldots ; k : t\ \} & \text{(properties)} \\
k ::= x \mid \ell & \text{(key)} \\
s ::= v \mid \mathrm{M}[t, \ldots, t] \mid \ell \mid c[b, \ldots, b] & \text{(simple)} \\
b ::= t \mid v \cdots v\,.\,t & \text{(binder)}
\end{array}
$$

(using just the meta-symbols ::= and |). Terms always allow an optional *properties* prefix p, which contains key-value pairs where the key can be a variable or a literal. A term can be a variable, a *meta-application* $\mathrm{M}[t_1, \ldots, t_n]$ or a *construction* $c[b_1, \ldots, b_n]$, where constructions differ from most other higher-order formalisms in that the only place where binders are allowed is in the b constructor arguments; these can take the form $v_1 \cdots v_n\,.\,t$ to bind $v_1, \ldots, v_n$ in the t subterm.[2] The only difference with actual CRSX syntax is that fonts are significant: we use META_n and **Primitive**, where actual CRSX syntax has `#meta_n` and `$Primitive`, respectively. In addition, like CRSX, we permit sort declarations as shown below and write lists of values in ";-terminated" form, *i.e.*, () is the empty list and (1; 2; 3;) the list of the three integer literals 1, 2, and 3, which is right recursive, so (1; 2;) = (1; (2; ())).

2.2 Example. Consider the following simple CRSX sorts.

N ::= Z | S[N] ; //usual Peano natural number sort
OP ::= + | − | × ; //arithmetic operation sort

E ::= Op[OP, E, E] | Nat[N] ; //expression sort
C ::= **List**[INS] ; //code sort is list of stack instructions (INS; ...INS;)
INS ::= OP[OP] | NAT[N] ; //stack instruction sort

With these we can define a translation scheme T with two arguments: an expression E and the "trailing" code fragment C.

T[E, C] :: C ; //compilation scheme from expression (and tail code) to code.
T[Nat[N], C] → (NAT[N]; C) ;
T[Op[OP, E_1, E_2], C] → T[E_1, T[E_2, (OP[OP]; C)]] ;

The compiler translates simple Peano arithmetic expressions to equivalent stack code: rewriting the term T[E, ()] will rewrite to the stack code for the expression E. For example,

T[Op[×, Nat[S[Z]], Op[+, Nat[Z], Nat[S[S[Z]]]]], ()]
→ T[Nat[S[Z]], T[Op[+, Nat[Z], Nat[S[S[Z]]]], (OP[×];)]]
→ (NAT[S[Z]]; T[Nat[], T[Nat[S[S[Z]]], (OP[+]; OP[×];)]])
→ (NAT[S[Z]]; NAT[Z]; NAT[S[S[Z]]]; OP[+]; OP[×];)

as expected. At this point, the evaluator of the stack computation takes over, and computation of the stack can be specified. The rules for the T scheme is a *constructor system*, with T the only function symbol and all other (including the hidden "nil" and "cons" of lists) as data constructors. Thus the system is trivially orthogonal and thus confluent.

[2] As usual when dealing with higher order terms with binders, the syntax really just defines "pre-terms" and we have to specify the renaming (α-)equivalence classes of terms, however, this introduces no surprises.

2.3 Example. Add the following optimization rules to Example 2.2:

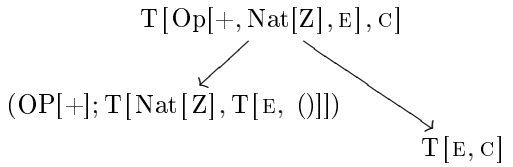
$\mathrm{Op}[+, \mathrm{Nat}[\mathrm{Z}], \mathrm{N}] \to \mathrm{N}$;
$\mathrm{Op}[\times, \mathrm{Nat}[\mathrm{S}[\mathrm{Z}]], \mathrm{N}] \to \mathrm{N}$;

Then the system is no longer confluent as we also have

$\mathrm{T}[\mathrm{Op}[\times, \mathrm{Nat}[\mathrm{S}[\mathrm{Z}]], \mathrm{Op}[+, \mathrm{Nat}[\mathrm{Z}], \mathrm{Nat}[\mathrm{S}[\mathrm{S}[\mathrm{Z}]]]]], ()]$
$\to \mathrm{T}[\mathrm{Op}[+, \mathrm{Nat}[\mathrm{S}[\mathrm{S}[\mathrm{Z}]]], \mathrm{Nat}[\mathrm{Z}]], ()]$
$\to \mathrm{T}[\mathrm{Nat}[\mathrm{S}[\mathrm{S}[\mathrm{Z}]]], ()]$
$\to \mathrm{NAT}[\mathrm{S}[\mathrm{S}[\mathrm{Z}]]]$

Clearly (from a compiler standpoint) a better result for run-time, and an equivalent result modulo the (runtime) evaluation rules for the stack code.

To proceed with completion, we need to know how to orient the critical pairs. Here is one critical pair:



It turns out that optimizations of this kind are *between data terms* and thus this allows us to pick the direction for the critical pair: essentially all optimizations "win" by giving priority to the data symbols; this works simply because using function rules proceed towards the end result whereas the data rules rewrite the source code, providing opportunities for improvement. Since all overlaps will be between an (original) function rule and one of the added rules that are temporarily demoting the system to a non-constructor system. If completion proceeds, we get the following replacement rules for $\mathrm{T}[\mathrm{Op}[\ldots]\ldots]$:

$\mathrm{T}[\mathrm{Op}[+, \mathrm{Nat}[\mathrm{Z}], \mathrm{E}_2], \mathrm{C}] \to \mathrm{T}[\mathrm{E}_2, \mathrm{C}]$;
$\mathrm{T}[\mathrm{Op}[+, \mathrm{Nat}[\mathrm{S}[\mathrm{N}_1]], \mathrm{E}_2], \mathrm{C}] \to \mathrm{T}[\mathrm{Nat}[\mathrm{S}[\mathrm{N}_1]], \mathrm{T}[\mathrm{E}_2, (\mathrm{OP}[+]; \mathrm{C})]]$;

$\mathrm{T}[\mathrm{Op}[-, \mathrm{E}_1, \mathrm{E}_2], \mathrm{C}] \to \mathrm{T}[\mathrm{E}_1, \mathrm{T}[\mathrm{E}_2, (\mathrm{OP}[-]; \mathrm{C})]]$;

$\mathrm{T}[\mathrm{Op}[\times, \mathrm{Nat}[\mathrm{Z}], \mathrm{E}_2], \mathrm{C}] \to \mathrm{T}[\mathrm{E}_2, \mathrm{C}]$;
$\mathrm{T}[\mathrm{Op}[\times, \mathrm{Nat}[\mathrm{S}[\mathrm{N}_1]], \mathrm{E}_2], \mathrm{C}] \to \mathrm{T}[\mathrm{Nat}[\mathrm{S}[\mathrm{N}_1]], \mathrm{T}[\mathrm{E}_2, (\mathrm{OP}[\times]; \mathrm{C})]]$;
$\mathrm{T}[\mathrm{Op}[\times, \mathrm{Nat}[\mathrm{S}[\mathrm{S}[\mathrm{N}_1]]], \mathrm{E}_2], \mathrm{C}] \to \mathrm{T}[\mathrm{Nat}[\mathrm{S}[\mathrm{S}[\mathrm{N}_1]]], \mathrm{T}[\mathrm{E}_2, (\mathrm{OP}[\times]; \mathrm{C})]]$;

The next example uses property sets, including the extensions for patterns.

2.4 Example. We'll consider a CRSX system for translation of list functions to a minimal assembly language. First the data sorts declaration for input expressions and output assembly code:

// Expressions.
$\mathrm{E} ::= v \mid \mathrm{Int}[\textbf{integer}] \mid \mathrm{Str}[\textbf{String}] \mid \mathrm{Nil} \mid \mathrm{Seq}[\mathrm{E}, \mathrm{E}] \mid \mathrm{IfNil}[\mathrm{E}, \mathrm{E}, \mathrm{E}] \mid \mathrm{Let}[\mathrm{E}, \mathrm{E}.\mathrm{E}]$;

// Assembly code.
$\{\mathrm{E}\colon \mathrm{AR}\}\mathrm{A} ::= \textbf{List}[\mathrm{AI}]$;
$\mathrm{AI} ::= \mathrm{L}[\mathrm{AL}] \mid \mathrm{J}[\mathrm{AL}] \mid \mathrm{PU} \mid \mathrm{POJN}[\mathrm{AL}] \mid \mathrm{POR}[\mathrm{AR}] \mid \mathrm{SR}[\mathrm{AR}] \mid \mathrm{SI}[\textbf{integer}] \mid \mathrm{SS}[\textbf{String}]$;
$\mathrm{AL} ::= \textit{label}$;
$\mathrm{AR} ::= \textit{register}$;

The assembly language instructions are meant to be Label, Jump, PUsh, POpJumpNil, POpRegister, SendRegister, SendInteger, and SendString; the idea with the assembly language is that the Push instruction pushes a new empty sequence context and that Send instructions send a value to the sequence at the top of the stack, which is tested or stored when pop'd. The declarations also establish that we use the built-in lists to compose assembly code, and that both labels and registers are represented using free variables.

2.5 Example. We will compile expressions to assembly with rules for a T translation scheme, as before, where the initial expression is meant to be something like (PU; $\{in : r_{\mathrm{i}};\ out : r_{\mathrm{o}}\}$ T[E, (POR[r_{o}];)]) .

$\{\mathrm{S}\,?\,v{:}\,r\}\mathrm{T}[v, \mathrm{A}] \to (\mathrm{SR}[r];\ \mathrm{A})$;
$\{\mathrm{S}\,?\,\neg\, v\}\mathrm{T}[v, \mathrm{A}] \to (\mathrm{SS}[\texttt{"Error"}];\ \mathrm{A})$;

The two rules express the translation rules for a variable v. In the first rule the environment S is constrained to contain a mapping from v to a register r, and in that case the assembly code is merely to send the register value to the stack top sequence. If the variable is not mapped as a key in the environment, as captured by the rule with a $\neg\, v$, then the result is the string `"Error"`.

To translate sequences we echo the computed values to the stack top sequence:

$\{\mathrm{S}\}\mathrm{T}[\mathrm{Nil}, \mathrm{A}] \to \mathrm{A}$;
$\{\mathrm{S}\}\mathrm{T}[\mathrm{Seq}[\mathrm{E}_1, \mathrm{E}_2], \mathrm{A}] \to \{\mathrm{S}\}\mathrm{T}[\mathrm{E}_1,\ \{\mathrm{S}\}\mathrm{T}[\mathrm{E}_2, \mathrm{A}]]$;

The rule for conditionals introduce "fresh" free variables into the term for the needed labels f and *fi*, which must be declared. A fresh variable is a usual variable introduced without a binding as a globally unique variable; since there is no binder it can never be substituted. Otherwise the test is merely whether the generated stack top sequence is empty with the usual jumps and labels to select the appropriate code and continue with the trailing code afterwards.

$-[\mathrm{Fresh}[f, \mathit{fi}]]$:
$\{\mathrm{S}\}\mathrm{T}[\mathrm{If}[\mathrm{E}, \mathrm{E}_1, \mathrm{E}_2], \mathrm{A}]$
$\to (\mathrm{PU};\ \{\mathrm{S}\}\mathrm{T}[\mathrm{E},\ (\mathrm{POJN}[f];\ \{\mathrm{S}\}\mathrm{T}[\mathrm{E}_1,\ (\mathrm{J}[\mathit{fi}];\ \mathrm{L}[f];\ \{\mathrm{S}\}\mathrm{T}[\mathrm{E}_2,\ (\mathrm{L}[\mathit{fi}];\ \mathrm{A})])])])$;

Finally, the rule for local binding:

$-[\mathrm{Fresh}[x, r]]$:
$\{\mathrm{S}\}\mathrm{T}[\mathrm{Let}[\mathrm{E}_1, v_1.\mathrm{E}_2[v_1]], \mathrm{A}] \to (\mathrm{PU};\ \{\mathrm{S}\}\mathrm{T}[\mathrm{E}_1,\ (\mathrm{POR}[r];\ \{\mathrm{S}\,!\,x{:}\,r\}\mathrm{T}[\mathrm{E}_2[x], \mathrm{A}])])$;

Here a fresh variable x is introduced as an E place holder for the computed values as well as a fresh register r for the actually computed value, and the environment that associates them is extended to map (the fresh copy of) the variable to the fresh register and then passed to the compilation of $\mathrm{E}_2[x]$.

2.6 Example. Completion examples grow more interesting when higher-order constructs are involved. Consider these two optimizations of the Let construct:

$\mathrm{If}[\mathrm{Nil}, \mathrm{E}_1, \mathrm{E}_2] \to \mathrm{E}_2$;
$\mathrm{Let}[\mathrm{E}, x.x] \to \mathrm{E}$;
$\mathrm{Let}[\mathrm{E}_1, x_1.\mathrm{E}_2[]] \to \mathrm{E}_2$;

The first merely states that trivial Lets should be eliminated. The second, however, uses the occurs check feature (from CRS) that when a bound variable is *not* mentioned in the pattern meta-application, then it is not permitted, which corresponds to eliminating "dead" Let binders.

Adding these two rules completes to the following, which, in addition to $\not\equiv$ uses a special $!\,v_1$ pattern substitute, which denotes that the variable v_1 *must* occur inside the term that matches $\text{E}_2[\,!\,v_1]$ (as the complement to the pattern $\text{E}_2[\,]$).

$\{\text{S}\}\text{T}[\text{If}[\text{Nil}, \text{E}_1, \text{E}_2], \text{A}] \to \{\text{S}\}\text{T}[\text{E}_2, \text{A}]$
$-[\text{Fresh}[f, \mathit{fi}]]$:
$\{\text{S}\}\text{T}[\text{If}[\text{E} \not\equiv \text{Nil}, \text{E}_1, \text{E}_2], \text{A}]$
$\to (\text{PU};\ \{\text{S}\}\text{T}[\text{E},\ (\text{POJN}[f];\ \{\text{S}\}\text{T}[\text{E}_1,\ (\text{J}[\mathit{fi}];\ \text{L}[f];\ \{\text{S}\}\text{T}[\text{E}_2,\ (\text{L}[\mathit{fi}];\ \text{A})])])])$;

$\{\text{S}\}\text{T}[\text{Let}[\text{E}, x.x], \text{A}] \to \{\text{S}\}\text{T}[\text{E}, \text{A}]$;
$\{\text{S}\}\text{T}[\text{Let}[\text{E}_1, x_1.\text{E}_2[]], \text{A}] \to \text{E}_2, \text{A}]$;
$-[\text{Fresh}[x, r]]$:
$\{\text{S}\}\text{T}[\text{Let}[\text{E}_1, v_1.\text{E}_2[\,!\,v_1] \not\equiv v_1], \text{A}] \to (\text{PU};\ \{\text{S}\}\text{T}[\text{E}_1,\ (\text{POR}[r];\ \{\text{S}\,!\,x{:}\,r\}\text{T}[\text{E}_2[x], \text{A}])])$;

2.7 Example. The previous rules can, for example, be used for the following rewrite:

$(\text{PU};\ \{\mathit{in}{:}\,r_\text{i};\ \mathit{out}{:}\,r_\text{o}\}\text{T}[\text{Let}[\text{If}[\text{Nil},\ \text{Nil},\ \mathit{in}],\ x.x],\ (\text{POR}[r_\text{o}];)])$
$\to (\text{PU};\ \{\mathit{in}{:}\,r_\text{i};\ \mathit{out}{:}\,r_\text{o}\}\text{T}[\text{If}[\text{Nil},\ \text{Nil},\ \mathit{in}],\ (\text{POR}[r_\text{o}];)])$
$\to (\text{PU};\ \{\mathit{in}{:}\,r_\text{i};\ \mathit{out}{:}\,r_\text{o}\}\text{T}[\mathit{in},\ (\text{POR}[r_\text{o}];)])$
$\to (\text{PU};\ \text{SR}[r_\text{in}];\ \text{POR}[r_\text{o}];)$

(The program merely copies the input register to the output register.)

3 Conclusion

The examples show how completion is used to integrate optimizations into a compiler. Rather than conclude, I would like to open a discussion: can completion be suitably generalized to capture this case? Can the semantics of the target language, which we have not shown here, be used to direct the completion such that general equivalences can be exploited as optimizations?

Acknowledgements. The author is grateful to Cynthia Kop for implementing the bulk of the completion procedure in CRSX while visiting Watson in 2011, enabling experiments like I describe here also for real compilers with thousands of rules. Finally thanks to the referees for several helpful suggestions.

References

[1] Peter Aczel. A general Church-Rosser theorem. http://www.ens-lyon.fr/LIP/REWRITING/MISC/AGeneralChurch-RosserTheorem.pdf, July 1978. Corrections at http://www.ens-lyon.fr/LIP/REWRITING/MISC/AGRT_corrections.pdf.

[2] Jan Willem Klop. *Combinatory Reduction Systems.* PhD thesis, University of Utrecht, 1980. Also available as Mathematical Centre Tracts 127.

[3] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

[4] Kristoffer H. Rose. Higher-order rewriting for executable compiler specifications. In *Proc. 5th HOR*, volume 49 of *Electronic Proceedings in Theoretical Computer Science*, pages 31–45, 2010.

[5] Kristoffer H. Rose. CRSX – combinatory reduction systems with extensions. In *Proc. 22nd RTA*, Leibniz International Proceedings in Informatics, pages 81–90, 2011. CRSX system available from http://crsx.sf.net.

Recording Completion for Finding and Certifying Proofs in Equational Logic*

Thomas Sternagel, René Thiemann, Harald Zankl
Institute of Computer Science
University of Innsbruck, Austria

Christian Sternagel
School of Information Science
JAIST, Japan

1 Introduction

Solving the word problem requires to decide whether an equation $s \approx t$ follows from an equational system (ES) $\mathcal{E}$. By Birkhoff's theorem this is equivalent to the existence of a conversion $s \leftrightarrow_{\mathcal{E}}^* t$. Knuth-Bendix completion [5] (if successful) gives a decision procedure: If an ES $\mathcal{E}$ is transformed into an equivalent convergent term rewrite system (TRS) $\mathcal{R}$, then $s \leftrightarrow_{\mathcal{E}}^* t$ iff the $\mathcal{R}$-normal forms of s and t coincide. (Note that completion does not construct such conversions explicitly.)

Example 1. For $\mathcal{E} = \{\mathsf{ff} \approx \mathsf{f}, \mathsf{ggf} \approx \mathsf{g}\}$ (where f and g are unary function symbols, for which we find it convenient to abbreviate $\mathsf{f}(\mathsf{g}(\mathsf{f}(x)))$ to fgf etc.) a possible choice of $\mathcal{R}$ is $\{\mathsf{ff} \to \mathsf{f}, \mathsf{gf} \to \mathsf{g}, \mathsf{gg} \to \mathsf{g}\}$. Since $\mathsf{fgf} \to_{\mathcal{R}}^* \mathsf{fg} \,{}_{\mathcal{R}}^*\!\leftarrow \mathsf{fgg}$, we have that $\mathsf{fgf} \approx \mathsf{fgg}$ follows from $\mathcal{E}$.

When we want to answer/certify whether $s \leftrightarrow_{\mathcal{E}}^* t$, we face the following situation: (1) It is hard to find a conversion but easy to certify a given one. (2) Under the assumption that Knuth-Bendix completion is successful, it is easy to decide the existence of a conversion (just rewrite s and t to $\mathcal{R}$-normal forms) but hard to certify this decision (e.g., by certifying that $\mathcal{E}$ and $\mathcal{R}$ are equivalent).

In this paper we introduce *recording completion*, which overcomes both problems. Recording completion keeps a history that allows us to reconstruct how the rules in $\mathcal{R}$ have been derived from the equations in $\mathcal{E}$. Then, from a join $s \to_{\mathcal{R}}^* \cdot \,{}_{\mathcal{R}}^*\!\leftarrow t$ a conversion $s \leftrightarrow_{\mathcal{E}}^* t$ can be reconstructed. Furthermore, recording completion enables the certification of completion proofs, i.e., to check that $\mathcal{R}$ and $\mathcal{E}$ are equivalent. Using equivalence together with confluence and termination certificates, it is also possible to certify that a conversion $s \leftrightarrow_{\mathcal{E}}^* t$ does *not* exist.

In addition to formalizing all required theorems like the critical pair theorem and soundness of completion, we have proven two new results: For finite completion proofs, i.e., where the completion procedure stops successfully after a finite number of steps, the strict encompassment condition (in the collapse-rule of Figure 1) is not required. Moreover, an infinite set of variables is essential for the critical pair theorem as well as modularity of confluence [8].

2 Proof Construction via Recording Completion

We extend the inference rules of completion [1] by a *history component* which allows us to infer how rules in $\mathcal{R}$ have been derived from equations in $\mathcal{E}$. The construction of a conversion $s \leftrightarrow_{\mathcal{E}}^* t$ (if possible at all) is then executed in three phases: (**record**) The inference rules of recording completion (see Figure 1) are applied to the ES $\mathcal{E}$. Upon success, a convergent TRS $\mathcal{R}$ (equivalent to $\mathcal{E}$) and a history $\mathcal{H}$ (recording how the rules in $\mathcal{R}$ have been derived) are computed. (**compare**) If the previous phase is successful, the test for $s \to_{\mathcal{R}}^* \cdot \,{}_{\mathcal{R}}^*\!\leftarrow t$ is performed. (**recall**) If the previous phase is successful, we construct $s \leftrightarrow_{\mathcal{E}}^* t$ from $s \to_{\mathcal{R}}^* \cdot \,{}_{\mathcal{R}}^*\!\leftarrow t$.

*Supported by Austrian Science Fund (FWF): P22467, P22767, J3202, and a grant by Hypo Tirol Bank.

$$\frac{(\mathcal{E},\mathcal{R},\mathcal{H})}{(\mathcal{E}\cup\{m\colon s\approx t\},\mathcal{R},\mathcal{H}\cup\{m\colon s \overset{j}{\leftarrow} u \overset{k}{\rightarrow} t\})}\ (\mathsf{deduce}) \qquad \text{if } s\ {}_{\mathcal{R}}\overset{j}{\leftarrow} u \overset{k}{\rightarrow}_{\mathcal{R}} t$$

$$\frac{(\mathcal{E}\cup\{i\colon s\approx t\},\mathcal{R},\mathcal{H})}{(\mathcal{E},\mathcal{R}\cup\{i\colon s\to t\},\mathcal{H})}\ (\mathsf{orient_l}) \qquad \text{if } s > t$$

$$\frac{(\mathcal{E}\cup\{i\colon s\approx t\},\mathcal{R},\mathcal{H}\cup\{i\colon s \overset{j}{\circ_1} u \overset{k}{\circ_2} t\})}{(\mathcal{E},\mathcal{R}\cup\{i\colon t\to s\},\mathcal{H}\cup\{i\colon t\,(\overset{k}{\circ_2})^{-1}\,u\,(\overset{j}{\circ_1})^{-1}\,s\})}\ (\mathsf{orient_r}) \qquad \text{if } t > s$$

$$\frac{(\mathcal{E}\cup\{i\colon s\approx t\},\mathcal{R},\mathcal{H})}{(\mathcal{E}\cup\{m\colon u\approx t\},\mathcal{R},\mathcal{H}\cup\{m\colon u \overset{l}{\leftarrow} s \overset{i}{\rightarrow} t\})}\ (\mathsf{simplify_l}) \qquad \text{if } s \overset{l}{\rightarrow}_{\mathcal{R}} u$$

$$\frac{(\mathcal{E}\cup\{i\colon s\approx t\},\mathcal{R},\mathcal{H})}{(\mathcal{E}\cup\{m\colon s\approx u\},\mathcal{R},\mathcal{H}\cup\{m\colon s \overset{i}{\rightarrow} t \overset{l}{\rightarrow} u\})}\ (\mathsf{simplify_r}) \qquad \text{if } t \overset{l}{\rightarrow}_{\mathcal{R}} u$$

$$\frac{(\mathcal{E}\cup\{i\colon s\approx s\},\mathcal{R},\mathcal{H}\cup\{i\colon s\circ_1 v\circ_2 s\})}{(\mathcal{E},\mathcal{R},\mathcal{H})}\ (\mathsf{delete})$$

$$\frac{(\mathcal{E},\mathcal{R}\cup\{i\colon s\to t\},\mathcal{H})}{(\mathcal{E},\mathcal{R}\cup\{m\colon s\to u\},\mathcal{H}\cup\{m\colon s \overset{i}{\rightarrow} t \overset{j}{\rightarrow} u\})}\ (\mathsf{compose}) \qquad \text{if } t \overset{j}{\rightarrow}_{\mathcal{R}} u$$

$$\frac{(\mathcal{E},\mathcal{R}\cup\{i\colon s\to t\},\mathcal{H})}{(\mathcal{E}\cup\{m\colon u\approx t\},\mathcal{R},\mathcal{H}\cup\{m\colon u \overset{j}{\leftarrow} s \overset{i}{\rightarrow} t\})}\ (\mathsf{collapse}) \qquad \text{if } s \overset{j}{\rightarrow}_{\mathcal{R}} u$$

Figure 1: The inference rules of *recording completion*.

In the sequel we give more details for each of the phases.

Record. The *record* phase uses the inference rules from Figure 1 where every rule/equation is annotated by a unique index i. Here, $\overset{i}{\rightarrow}_{\mathcal{R}}$ denotes an $\mathcal{R}$-reduction using the rule with index i. The inference rules are similar to the standard rules except for the following two differences: In the collapse-rule we dropped the condition of strict encompassment. Since we only consider finite runs, this condition is no longer required for soundness (cf. Theorem 1). Furthermore, there is a new history component $\mathcal{H}$ whose entries are of the form $i : s \overset{j}{\circ_1} u \overset{k}{\circ_2} t$ where i is the index of the entry, j and k are indices of equations/rules, s, u, and t are terms, and $\circ_1, \circ_2 \in \{\leftarrow, \rightarrow, \approx\}$.

Let us take a closer look at the extended inference rules. For deduce the peak $s \overset{j}{\leftarrow}_{\mathcal{R}} u \overset{k}{\rightarrow}_{\mathcal{R}} t$ that triggers the new equation $s \approx t$ is stored in a history entry (where m is assumed to be a fresh index that is larger than every earlier index). By $\mathsf{orient_l}$ we orient an equation from left to right and the corresponding history entry remains unchanged, whereas by $\mathsf{orient_r}$ we orient an equation from right to left and thus have to "mirror" the corresponding history entry. Here $>$ is a reduction order, which is part of the input. The rules $\mathsf{simplify_l}$ and $\mathsf{simplify_r}$ are used to $\mathcal{R}$-rewrite a left- or right-hand side of an equation. With delete we remove trivial equations from $\mathcal{E}$ and the corresponding history entry from $\mathcal{H}$. Finally, compose rewrites a right-hand side of a rule in $\mathcal{R}$ while collapse does the same for left-hand sides.

We write $(\mathcal{E}_i,\mathcal{R}_i,\mathcal{H}_i) \rightsquigarrow (\mathcal{E}_{i+1},\mathcal{R}_{i+1},\mathcal{H}_{i+1})$ for the application of an arbitrary inference rule to the triple $(\mathcal{E}_i,\mathcal{R}_i,\mathcal{H}_i)$ resulting in $(\mathcal{E}_{i+1},\mathcal{R}_{i+1},\mathcal{H}_{i+1})$.

Definition 1. A *run* of recording completion for $\mathcal{E}$ is a finite sequence $(\mathcal{E}_0,\mathcal{R}_0,\mathcal{H}_0) \rightsquigarrow^n (\mathcal{E}_n,\mathcal{R}_n,\mathcal{H}_n)$ of rule applications, where $\mathcal{E}_0 = \{i : s \approx t \mid s \approx t \in \mathcal{E}\}$ with fresh index i for each equation, $\mathcal{R}_0 = \varnothing$, and the initial history is $\mathcal{H}_0 = \{i : s \overset{i}{\rightarrow} t \approx t \mid i : s \approx t \in \mathcal{E}_0\}$. A

	$\mathcal{E}_0$	$\mathcal{R}_0$		$\mathcal{H}_0$
1:	$\mathsf{ff} \approx \mathsf{f}$	$\varnothing$	1:	$\mathsf{ff} \overset{1}{\to} \mathsf{f} \approx \mathsf{f}$
2:	$\mathsf{ggf} \approx \mathsf{g}$		2:	$\mathsf{ggf} \overset{2}{\to} \mathsf{g} \approx \mathsf{g}$

(a) Initial state.

$\mathcal{E}_n$		$\mathcal{R}_n$		$\mathcal{H}_n$
$\varnothing$	1:	$\mathsf{ff} \to \mathsf{f}$	1:	$\mathsf{ff} \overset{1}{\to} \mathsf{f} \overset{0}{\approx} \mathsf{f}$
	4:	$\mathsf{gf} \to \mathsf{g}$	2:	$\mathsf{ggf} \overset{2}{\to} \mathsf{g} \overset{0}{\approx} \mathsf{g}$
	5:	$\mathsf{gg} \to \mathsf{g}$	3:	$\mathsf{ggf} \overset{1}{\leftarrow} \mathsf{ggff} \overset{2}{\to} \mathsf{gf}$
			4:	$\mathsf{gf} \overset{3}{\leftarrow} \mathsf{ggf} \overset{2}{\to} \mathsf{g}$
			5:	$\mathsf{gg} \overset{4}{\leftarrow} \mathsf{ggf} \overset{2}{\to} \mathsf{g}$

(b) Final state.

Table 2: Example of recording completion.

run is *successful* if $\mathcal{E}_n = \varnothing$ and all critical pairs of $\mathcal{R}_n$ that are not contained in $\bigcup_{i \le n} \mathcal{E}_i$ are joinable by $\mathcal{R}_n$. A run is *sound* if $\mathcal{R}_n$ is convergent and equivalent to $\mathcal{E}_0$.

The requirement on critical pairs for a successful run can be replaced by local confluence of $\mathcal{R}_n$.

Example 2. Recall $\mathcal{E}$ from Example 1. We start with the triple depicted in Table 2(a) and perform recording completion. Note that LPO with empty precedence orients all emerging rules in the desired direction. After orienting rules 1 and 2 from left to right we deduce a critical pair between rules 2 and 1, resulting in the equation $3 : \mathsf{ggf} \approx \mathsf{gf}$ and the history entry $3 : \mathsf{ggf} \overset{1}{\leftarrow} \mathsf{ggff} \overset{2}{\to} \mathsf{gf}$. Next we simplify the left-hand side of equation 3 by an application of rule 2 and obtain the equation $4 : \mathsf{g} \approx \mathsf{gf}$ with corresponding history entry $4 : \mathsf{g} \overset{2}{\leftarrow} \mathsf{ggf} \overset{3}{\to} \mathsf{gf}$. Orienting rule 4 from right to left causes the history entry to be mirrored, i.e., $4 : \mathsf{gf} \overset{3}{\leftarrow} \mathsf{ggf} \overset{2}{\to} \mathsf{g}$. Rules 2 and 4 allow to deduce equation $5 : \mathsf{gg} \approx \mathsf{g}$ with history $5 : \mathsf{gg} \overset{4}{\leftarrow} \mathsf{ggf} \overset{2}{\to} \mathsf{g}$, which we orient from left to right. Collapsing the left-hand side of rule 2 with rule 5 yields $6 : \mathsf{gf} \approx \mathsf{g}$ with $6 : \mathsf{gf} \overset{5}{\leftarrow} \mathsf{ggf} \overset{2}{\to} \mathsf{g}$. Now rule 4 simplifies equation 6 into $7 : \mathsf{g} \approx \mathsf{g}$ with $7 : \mathsf{g} \overset{4}{\leftarrow} \mathsf{gf} \overset{6}{\to} \mathsf{g}$, which is immediately deleted afterwards. Finally, $\mathcal{E}_n$ is empty and as all remaining critical pairs of $\mathcal{R}_n$ are joinable, the procedure can be stopped. Since there is no rule with index 6, the history entry 6 can be dropped. Hence, we obtain the result depicted in Table 2(b) where $\mathcal{R}_n$ is convergent and equivalent to $\mathcal{E}_0$.

We have formalized soundness of recording completion in IsaFoR [7] (see `Completion.thy`).

Theorem 1. *Every successful run of recording completion is sound.* □

Compare. Let $(\mathcal{E}, \varnothing, \mathcal{H}_0) \rightsquigarrow^n (\varnothing, \mathcal{R}, \mathcal{H}_n)$ be a successful run of recording completion and $s \approx t$ an equation. In the *compare* phase we test joinability of the terms s and t with respect to $\mathcal{R}$. If the two terms are joinable, then $s \approx t$ follows from $\mathcal{E}$ and the next phase constructs an $\mathcal{E}$-conversion $s \leftrightarrow^*_{\mathcal{E}} t$. Otherwise, $s \not\approx t$ w.r.t. $\mathcal{E}$. The *compare* phase is sound (cf. Theorem 1).

Recall. Let $(\mathcal{E}, \varnothing, \mathcal{H}_0) \rightsquigarrow^n (\varnothing, \mathcal{R}, \mathcal{H}_n)$ be a run of recording completion. Then the *recall* phase transforms a join $s \to^*_{\mathcal{R}} \cdot \ ^*_{\mathcal{R}}\!\leftarrow t$ into a conversion $s \leftrightarrow^*_{\mathcal{E}} t$ as follows. For each step $t_1 \overset{i}{\to} t_2$ where the index i is not in $\mathcal{E}$ the corresponding history entry is *inserted*. Let $i : \ell \to r$ be the rule with index i. Then there must be a history entry $i : \ell \overset{j}{\circ}_1 u \overset{k}{\circ}_2 r$, a position p, and a substitution σ such that $t_1|_p = \ell\sigma$ and $t_2|_p = r\sigma$. The step $t_1 \overset{i}{\to} t_2$ is replaced by the conversion $t_1 \overset{j}{\circ}_1 t_1[u\sigma]_p \overset{k}{\circ}_2 t_2$. This process terminates since $i > j, k$, i.e., any history entry (not in $\mathcal{H}_0$) refers to smaller indices and finally we arrive at a conversion using indices from $\mathcal{E}$.

The next lemma states the desired property of the *recall* phase. Note that we do not need a *successful* run of recording completion but any join $s \to_{\mathcal{R}}^* \cdot {}_{\mathcal{R}}^*\!\leftarrow t$ is transformed into $s \leftrightarrow_{\mathcal{E}}^* t$.

Lemma 1. *Let $(\mathcal{E}, \varnothing, \mathcal{H}_0) \rightsquigarrow^n (\mathcal{E}_n, \mathcal{R}, \mathcal{H}_n)$ be a run of recording completion. Then the recall phase transforms any join using rules from $\mathcal{R}$ into a conversion using rules from $\mathcal{E}$.* □

Alternatively, one can ensure $\leftrightarrow_{\mathcal{R}}^* \subseteq \leftrightarrow_{\mathcal{E}}^*$ to derive $s \leftrightarrow_{\mathcal{E}}^* t$ from $s \to_{\mathcal{R}}^* \cdot {}_{\mathcal{R}}^*\!\leftarrow t$. The former can be established by showing that all history entries $i\colon s' \circ_1^j u \circ_2^k t'$ are consequences of $\mathcal{E}$ (i.e., $s' \leftrightarrow_{\mathcal{E}}^* t'$) and can thus be used as auxiliary equations. To avoid cyclic references, history entries are processed in order of their indices. This approach requires the certifier to support such auxiliary equations. In return, proofs become much shorter as the history itself is the proof of $\leftrightarrow_{\mathcal{R}}^* \subseteq \leftrightarrow_{\mathcal{E}}^*$ which obviously has linear size. In contrast, the recall phase might produce certificates where the conversion $s \leftrightarrow_{\mathcal{E}}^* t$ is exponentially larger than the join $s \to_{\mathcal{R}}^* \cdot {}_{\mathcal{R}}^*\!\leftarrow t$.

3 Formalization and Certification

To maximize the reliability of the computed results, we have developed a verified certifier using the proof assistant Isabelle/HOL. Based on IsaFoR [7] the code generation facilities of Isabelle/HOL allow to generate the verified certifier CeTA, which is able to certify or falsify conversions, completion proofs, and equational proofs and disproofs which are performed via completion.[1] For the latter, although Theorem 1 has been formalized, it is not checked whether the completion rules are applied correctly. Instead it is just verified if the result of the completion procedure is a convergent TRS equivalent to the initial set of equations.

To decide whether $s \leftrightarrow_{\mathcal{E}}^* t$ holds it suffices to find a convergent TRS $\mathcal{R}$ that is equivalent to $\mathcal{E}$ and decide whether the $\mathcal{R}$-normal forms of s and t coincide.

For equivalence of $\mathcal{R}$ and $\mathcal{E}$ we have to consider two directions. To decide $\leftrightarrow_{\mathcal{E}}^* \subseteq \leftrightarrow_{\mathcal{R}}^*$, by convergence of $\mathcal{R}$ we just have to check that for all $s \approx t \in \mathcal{E}$, the $\mathcal{R}$-normal forms of s and t coincide. For the other direction, $\leftrightarrow_{\mathcal{R}}^* \subseteq \leftrightarrow_{\mathcal{E}}^*$, we have to guarantee $\ell \leftrightarrow_{\mathcal{E}}^* r$ for all $\ell \to r \in \mathcal{R}$. Here, we use the information from recording completion to get the required derivations.

Hence, to certify that such a proof is correct we have to guarantee that $\mathcal{R}$ is convergent by showing termination and local confluence. Concerning termination, already several techniques have been formalized in IsaFoR. Hence, the new part is on the certification of local confluence. Here, the key technique is the critical pair theorem of Huet [3]—making a result by Knuth and Bendix [5] explicit. It states that $\mathcal{R}$ is locally confluent iff all critical pairs of $\mathcal{R}$ are joinable.

During the formalization we detected that in general (no assumption on the set of variables $\mathcal{V}$) there is a problem of renaming variables in rules for building critical pairs. To solve this problem without demanding an infinite set of variables, we see two alternatives: Either keep the set of variables and when building critical pairs try to rename variables apart as good as possible; or use an enlarged set of variables in the definition of critical pairs (so that there are enough variables to perform renamings). It turns out that for both alternatives the critical pair theorem does not hold.

For the first alternative it is easy to see that joinability of critical pairs does not imply local confluence. To this end, consider $\mathcal{V} = \{x\}$ and $\mathcal{R} = \{\mathsf{f}(\mathsf{a}, x) \to \mathsf{a}, \mathsf{f}(x, \mathsf{b}) \to \mathsf{b}\}$. This TRS is not locally confluent due to the peak $\mathsf{a} \leftarrow \mathsf{f}(\mathsf{a}, \mathsf{b}) \to \mathsf{b}$. But without changing $\mathcal{V}$ it is not possible to rename the variables of the two rules in $\mathcal{R}$ apart, such that their left-hand sides are unifiable. Hence, for the first alternative all critical pairs are joinable.

[1] Both IsaFoR and CeTA are freely available from http://cl-informatik.uibk.ac.at/software/ceta/.

$$\mathcal{R}_1$$
$$\mathsf{f}(\mathsf{g}(x_1,x_2),\mathsf{g}(x_3,x_4)) \to \mathsf{h}(x_1,\mathsf{h}(x_2,\mathsf{g}(x_3,x_4)))$$
$$\mathsf{f}(\mathsf{g}(\mathsf{g}(x_1,x_2),\mathsf{g}(x_3,x_4)),x_5) \to \mathsf{h}(x_5,\mathsf{h}(\mathsf{g}(x_1,x_2),\mathsf{g}(x_3,x_4)))$$

$\mathcal{R}_2$	$\mathcal{R}_3$	$\mathcal{R}_4$
$\mathsf{h}(\mathsf{g}(t,x_1),\mathsf{h}(x_2,x_3)) \to \mathsf{c}$	$\mathsf{h}(\mathsf{g}(y,x_1),\mathsf{h}(\mathsf{g}(x_2,x_3),\mathsf{g}(x_4,x_5))) \to \mathsf{c}$	$\mathsf{h}(x_1,\mathsf{c}) \to \mathsf{c}$
$\mathsf{h}(\mathsf{g}(x_1,t),\mathsf{h}(x_2,x_3)) \to \mathsf{c}$	$\mathsf{h}(\mathsf{g}(x_1,y),\mathsf{h}(\mathsf{g}(x_2,x_3),\mathsf{g}(x_4,x_5))) \to \mathsf{c}$	
$\mathsf{h}(x_1,\mathsf{h}(\mathsf{g}(t,x_2),x_3)) \to \mathsf{c}$	$\mathsf{h}(\mathsf{g}(x_1,x_2),\mathsf{h}(\mathsf{g}(y,x_3),\mathsf{g}(x_4,x_5))) \to \mathsf{c}$	
$\mathsf{h}(x_1,\mathsf{h}(\mathsf{g}(x_2,t),x_3)) \to \mathsf{c}$	$\mathsf{h}(\mathsf{g}(x_1,x_2),\mathsf{h}(\mathsf{g}(x_3,y),\mathsf{g}(x_4,x_5))) \to \mathsf{c}$	
$\mathsf{h}(x_1,\mathsf{h}(x_2,\mathsf{g}(t,x_3))) \to \mathsf{c}$	$\mathsf{h}(\mathsf{g}(x_1,x_2),\mathsf{h}(\mathsf{g}(x_3,x_4),\mathsf{g}(y,x_5))) \to \mathsf{c}$	
$\mathsf{h}(x_1,\mathsf{h}(x_2,\mathsf{g}(x_3,t))) \to \mathsf{c}$	$\mathsf{h}(\mathsf{g}(x_1,x_2),\mathsf{h}(\mathsf{g}(x_3,x_4),\mathsf{g}(x_5,y))) \to \mathsf{c}$	

Table 3: Rule schema for $\mathcal{R}_c$ with $y \in \mathcal{V}$ and $t \in \{\mathsf{c}, \mathsf{f}(x_4,x_5), \mathsf{g}(x_4,x_5), \mathsf{h}(x_4,x_5)\}$.

For the second alternative, $\mathcal{R}$ may be locally confluent although not every critical pair is joinable: the next example shows that if $\mathcal{V}$ is finite and $\mathcal{R}$ is locally confluent, then it need not be the case that all critical pairs of $\mathcal{R}$ are joinable.

Example 3. Let $\mathcal{R}_c = \bigcup_{i=1}^{4} \mathcal{R}_i$ be the TRS over $\mathcal{F} = \{\mathsf{f},\mathsf{g},\mathsf{h},\mathsf{c}\}$ and $\mathcal{V} = \{x_1,\ldots,x_5\}$ depicted in Table 3. It is constructed in such a way that each term $\mathsf{h}(\mathsf{g}(t_1,t_2),\mathsf{h}(\mathsf{g}(t_3,t_4),\mathsf{g}(t_5,t_6)))$ can be reduced to c (via $\mathcal{R}_2$ if some t_i is not a variable and via $\mathcal{R}_3$ if $t_i = t_j$ for $i < j$). Since there are only five different variables in $\mathcal{V}$, indeed every term $\mathsf{h}(\mathsf{g}(t_1,t_2),\mathsf{h}(\mathsf{g}(t_3,t_4),\mathsf{g}(t_5,t_6)))$ can be reduced to c. Moreover, all critical pairs, for which one of the rules is taken from $\mathcal{R}_c \setminus \mathcal{R}_1$, are joinable. Hence, the only critical pair that remains to be considered arises between the two rules of $\mathcal{R}_1$ where $u = \mathsf{f}(\mathsf{g}(\mathsf{g}(x_1,x_2),\mathsf{g}(x_3,x_4)),\mathsf{g}(x_5,x_6))$:

$$\mathsf{h}(\mathsf{g}(x_1,x_2),\mathsf{h}(\mathsf{g}(x_3,x_4),\mathsf{g}(x_5,x_6))) \leftarrow u \rightarrow \mathsf{h}(\mathsf{g}(x_5,x_6),\mathsf{h}(\mathsf{g}(x_1,x_2),\mathsf{g}(x_3,x_4)))$$

This critical pair is not joinable, as both terms are $\mathcal{R}_c$-normal forms. However, $\mathcal{R}_c$ is confluent since every *instance* of the critical pair (w.r.t. $\mathcal{T}(\mathcal{F},\mathcal{V})$) is joinable to c.

The example shows that confluence depends on the set of *variables* which most often is assumed to be infinite. Without this assumption, the requirement that all critical pairs have to be joinable can be too strict.[2] Another important consequence is that in the case of finite $\mathcal{V}$, Toyama's modularity result for confluence [8] does no longer hold.

Corollary 1. *Confluence is not a modular property of TRSs for an arbitrary set of variables.*

To summarize, it is not possible to formalize the critical pair theorem for arbitrary sets $\mathcal{V}$. Hence, we formalized it for strings, where it is conveniently possible to rename variables of rules apart without changing the type of variables (by using different prefixes). Of course, if $\mathcal{V}$ is infinite we can always obtain a renaming function (take *some* fresh variables) by the *Axiom of Choice*. However, then the definition of critical pairs is not executable.

Theorem 2. *A TRS over $\mathcal{T}(\mathcal{F}, String)$ is locally confluent iff all critical pairs are joinable.*

Note that the theorem does not require any variable-condition for $\mathcal{R}$. Hence, $\mathcal{R}$ may, e.g., contain left-hand sides which are variables or free variables in right-hand sides.

[2] We have only shown this result for $|\mathcal{V}| = 5$. However, $\mathcal{R}_c$ can be adapted to any finite $\mathcal{V}$ with $5 \leq |\mathcal{V}|$.

4 Implementation and Conclusion

We performed experiments[3] for completion proofs using KBCV [6] and MKBTT [9] (on 115 ESs).[4] Within a time limit of 300 seconds, KBCV could complete 86 ESs and MKBTT 80 ESs while both tools together succeeded on 94. The corresponding 94 completion proofs could be certified by CeTA (version 2.4). For an evaluation of other completion tools we refer to [4].

In our experiments we considered both possibilities (mentioned at the end of Section 2) to ensure $\leftrightarrow_{\mathcal{R}}^* \subseteq \leftrightarrow_{\mathcal{E}}^*$. While KBCV 1.6 performs the recall phase to explicitly construct $\ell \leftrightarrow_{\mathcal{E}}^* r$ for each $\ell \to r \in \mathcal{R}$, KBCV 1.7 just exports the relevant history entries, which are used as auxiliary equations. Hence it is not surprising that from the 86 ESs which KBCV 1.6 could complete only 80 have been certified. For two ESs (`TPTP_GRP487-1_theory` and `TPTP_GRP_490-1_theory`) the recall phase did not terminate within the time limit and for the remaining ESs (`LS94_P1`, `TPTP_GRP_481-1_theory`, `TPTP_GRP_486-1_theory`, `TPTP_GRP_490-1_theory`) the certificate was too large (365 MB, 230 MB, 406 MB, 581 MB) for CeTA. However, when using auxiliary equations all proofs could be computed and certified (typically within a second). Hence further optimization of the proof format seems dispensable.

While MKBTT follows recording completion, C*i*ME3 implements an annotated version of ordered completion [2]. Here—in contrast to our approach—the history is not saved as a standalone component but directly integrated into terms, equations, and rules. Hence a term t comes with an original version t^0, a current version t^*, and a reduction sequence from t^0 to t^*. Similarly an equation $s \approx t$ also contains all intermediate (rewrite) steps that show that both terms are equal. It requires further investigations to evaluate the pros and cons of the two approaches.

Acknowledgments: We thank Sarah Winkler for integrating certifiable output into MKBTT and helpful discussion.

References

[1] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge Univ. Press, 1999.

[2] Évelyne Contejean and Pierre Corbineau. Reflecting proofs in first-order logic with equality. In *CADE 2005*, volume 3632 of *LNAI*, pages 7–22, 2005.

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

[4] Dominik Klein and Nao Hirokawa. Maximal completion. In *RTA 2011*, volume 10 of *LIPIcs*, pages 71–80, 2011.

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

[6] Thomas Sternagel and Harald Zankl. KBCV – Knuth-Bendix completion visualizer. In *IJCAR 2012*, LNAI, 2012. To appear.

[7] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In *TPHOLs 2009*, volume 5674 of *LNCS*, pages 452–468, 2009.

[8] Yoshihito Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.

[9] Sarah Winkler, Haruhiko Sato, Aart Middeldorp, and Masahito Kurihara. Multi-completion with termination tools. *Journal of Automated Reasoning*. doi:10.1007/s10817-012-9249-2. To appear.

[3] http://cl-informatik.uibk.ac.at/software/kbcv/experiments/12iwc
[4] http://cl-informatik.uibk.ac.at/software/mkbtt

Automatically Finding Non-confluent Examples in Abstract Rewriting

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

Abstract

We present a technique based on SAT solving to find counter examples in abstract rewriting fully automatically, in particular satisfying non-confluence.

1 Introduction

Rewriting notions like termination, normal forms, and confluence can be described in an abstract way referring to rewriting only by a binary relation. Several theorems on rewriting, like Newman's lemma and Van Oostrom's decreasing diagrams [4, 5] can be proved in this abstract setting. For investigating possible modifications of such theorems, it is fruitful to be able to find examples combining a given list of properties.

As an example, consider the following question. Can we find three rewriting systems for which the union of every two is complete, but for which the union of all three is not confluent? Here a rewriting system is called *complete* if it is both confluent and terminating.

In this note we describe a technique by which such an example is found fully automatically. In this question we look for a set A of objects to be rewritten, and a couple of binary relations on A that describe rewrite steps and satisfy the given properties. We focus on the situation where the set A is finite. In applications of rewriting, most time the sets of objects are infinite. But in case some property does not hold, often this can be shown by a finite counter example, and often the smallest possible counter example gives the best insight why the property does not hold. So we will not only focus on counter examples in which A is finite, we will also focus on counter examples in which $n = \#A$ is as small as possible.

As a binary relation on a set of n elements can be expressed by n^2 Boolean variables, our problem area can be seen as a class of constraint problems on Boolean variables. Our goal is to express rewriting properties like termination and confluence in propositional logic in such a way that we may express our problems as propositional SAT(isfiability) problems, by which we may exploit current powerful SAT solvers.

We succeeded in expressing all abstract rewriting properties of our interest. For a full overview we refer to the paper [6]; in this note we focus on completeness and confluence.

Based on these characterizations we developed our tool `Carpa` (Counter examples for Abstract Rewriting Produced Automatically). In this tool a list of desired properties for a number of binary relations can be entered, and then for a given number $n = \#A$ it either gives an example in which the desired properties hold, or concludes that such an example does not exist. Internally this tool first builds a formula expressing the desired properties based on the characterizations presented in this note, then it calls an external SAT solver. In case the SAT solver concludes that the formula is unsatisfiable, the tool concludes that there is no solution, and in case the SAT solver concludes that the formula is satisfiable, the tool investigates the corresponding satisfying assignment and extracts an example out of it that satisfies the given properties. Although internally this SAT solving is crucial, the user of `Carpa` does not see this and only sees the automatic creation of an example of a set of binary relations that satisfies the given list of desired properties.

2 Encoding of Properties

In this section we summarize the characterizations of the main properties used to encode them in SAT formulas. For more explanation, examples and proofs we refer to [6].

Theorem 1. *A binary relation R on a finite set is terminating if and only if a binary relation S on the same set exists that is transitive and irreflexive, and for which $R \subseteq S$.*

Using this theorem the requirement of termination of a relation R can be expressed in a SAT formula: introduce an auxiliary relation S and express $R \subseteq S$ and transitivity and irreflexivity of S.

Theorem 2. *Let R be a relation on a finite set A and let $k \geq 1$. Let R_i be relations on A for $i = 1, 2, \ldots, k$, satisfying*

$$R_1 = I \cup R \cup R^2, \quad \text{and} \quad R_{i+1} = R_i \cup R_i^2 \quad \text{for } i = 1, \ldots, k-1.$$

Assume that either $2^k \geq \#A$ or R_k is transitive. Then $R_k = R^*$.

So for expressing the relation R^* for a given binary relation R choose a small k satisfying $2^k \geq \#A$ and introduce auxiliary relations $R_1, R_2, \ldots, R_k$ and create formulas expressing $R_1 = I \cup R \cup R^2$ for I being the identity, and $R_{i+1} = R_i \cup R_i^2$ for $i = 1, \ldots, k-1$. Then R_k describes the desired relation R^*.

The bound 2^k in Theorem 2 is optimal: for $A = \{1, 2, \ldots, 2^k + 1\}$ and $R = \{(1,2), (2,3), \ldots, (2^k, 2^k+1), (2^k+1, 1)\}$ we have $(1,1) \in R^+$, but $(1,1) \notin R_k$ since 1 can not be reached from 1 in less than $2^k + 1$ steps.

Using Theorem 2 we can express confluence and local confluence as follows.

For two binary relations R and S on a set A we write $\mathsf{peak}(R, S)$ for $R^{-1} \cdot S$, so

$$(x, y) \in \mathsf{peak}(R, S) \iff \bigvee_{z \in A} (zRx \wedge zSy).$$

Similarly, we write $\mathsf{valley}(R, S)$ for $R \cdot S^{-1}$, so

$$(x, y) \in \mathsf{valley}(R, S) \iff \bigvee_{z \in A} (xRz \wedge ySz).$$

Now by definition a relation R is confluent if and only if

$$\mathsf{peak}(R^*, R^*) \subseteq \mathsf{valley}(R^*, R^*),$$

and a relation R is locally confluent if and only if

$$\mathsf{peak}(R, R) \subseteq \mathsf{valley}(R^*, R^*).$$

These properties are expressed in propositional logic in a straightforward way.

Also weak normalization (every element has at least one normal form) and the unique normal form property (every element has at most one normal form) for a binary relation R can be expressed when R^* is available. First we specify a normal form to be an element x such that $(x, y) \notin R$ for all y. Now R is weakly normalizing if for every x a normal form y exists such that $(x, y) \in R^*$, and R has the unique normal form property if for every x and every two distinct normal forms y, z the property $(x, y) \in R^* \wedge (x, z) \in R^*$ does not hold. Again these properties are easily expressed in propositional logic.

A simple way to express completeness is specify both termination and confluence in the way just described. However, the following characterization is often more efficient, for instance for Example 5, both in the size of the formulas and in the time required by the SAT solver.

Theorem 3. *A binary relation R on a set A is complete if and only if two binary relations S and T on A exist such that the following properties hold:*

1. $R \subseteq S$,
2. *S is transitive and irreflexive,*
3. $\bigwedge_{x\in A}(xTx \vee \bigvee_{y\in A} xRy)$,
4. $\bigwedge_{x,y\in A}((xSy \wedge yTy) \rightarrow xTy)$,
5. $\bigwedge_{x,y,z\in A, y\neq z} \neg(xTy \wedge xTz)$.

For a complete relation R it is easy to check that $S = R^+$ and T defined by

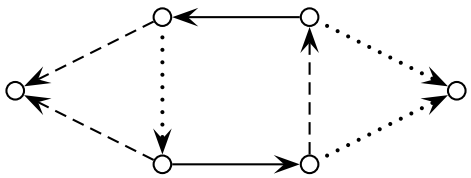
$$T = \{(x, y) \mid x \in A \text{ and } y \text{ is the normal form of } x\}$$

satisfy all these requirements.

Using Theorem 3 the following approach specifies a relation R to be complete in a SAT formula: introduce two fresh relations S and T and add the requirements of Theorem 3 to the SAT formula.

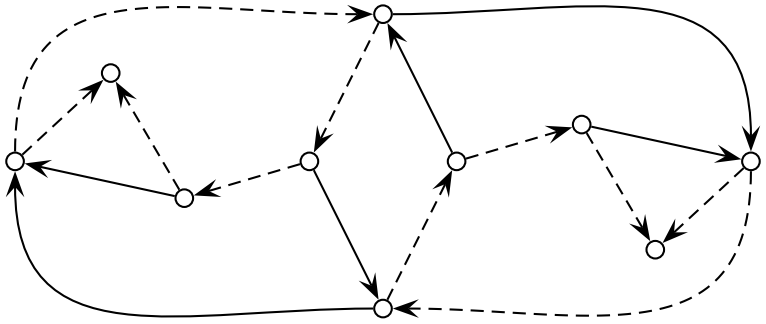
We give a few examples exploiting this approach. The first one solves the question posed in the introduction.

Example 4. If we have a set of binary relations of which the union of any two is complete, then it is easily seen that the union of all of them is locally confluent, as the union of any two is locally confluent. But can we conclude confluence of this union? This turns out to be not the case: we will give an example of three binary relations for which the union of any two of them is complete, but for which the union of all three is not confluent. The requirements are expressed in a formula: for the union of any two we specify completeness according to the requirements in Theorem 3; for non-confluence we specify that for the transitive reflexive closure of the union of all three the peak is not a subset of the valley. It turns out that this formula is unsatisfiable for $n = \#A \leq 5$, but for $n = 6$ it is satisfiable, and the corresponding satisfying assignment yields



in which the three relations are indicated by solid, dashed and dotted arrows, respectively.

Example 5. Let R and S be two complete binary relations satisfying $R^{-1} \cdot S \subseteq (S \cup R^*) \cdot (R^{-1})^*$. Can we conclude that the union is confluent? This question arose in investigations of simply typed lambda calculus, see [2, 3]. The answer is negative as is shown by the following example in which R steps are denoted by solid arrows and S steps are denoted by dashed arrows.



Using the representation of completeness based on Theorem 3 it turned out that a solution could be found within seconds, for several variants of the specification of the problem. All solutions that we found on 10 elements turned out to coincide with the example given above, sometimes after removing redundant arrows.

This number of 10 elements is minimal when requiring two distinct normal forms as in the picture. After a few hours of computation the corresponding formula for $n = 9$ could be proved to be unsatisfiable.

3 Implementation

We made an implementation called `Carpa` (Counter examples for Abstract Rewriting Produced Automatically) for entering a list of properties of binary relations. `Carpa`then either builds a set of binary relations on the specified number of elements that satisfies these properties, or shows that this is impossible. The tool `Carpa` can be downloaded from

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

Internally `Carpa` does the job via SAT solving and the techniques described in this note. As the SAT solver it uses `Yices` [1], which is not only a SAT solver, but also an SMT solver (satisfiability modulo theories). We also developed a version of our tool generating the formulas in dimacs format and then calling the SAT solver `Minisat` instead. As the version generating formulas in SMT format and calling `Yices` turned out to be the most efficient for the examples requiring one second or more, we decided only to distribute this one.

We defined an input format in which all properties discussed in this note can be specified directly, abstracting from the auxiliary relations that have to be introduced internally. In particular:

- `subs`, where `subs`(R, S) means that $R \subseteq S$,
- `sn`, where `sn`(R) means that R is terminating,
- `wn`, where `wn`(R) means that R is weakly normalizing,
- `cr`, where `cr`(R) means that R is confluent,
- `wcr`, where `wcr`(R) means that R is locally confluent,
- `un`, where `un`(R) means that R has the unique normal form property.

For all of these properties the negation can be specified by preceding the name by `n`, for instance, `ncr`(R) means that R is not confluent. Internally, the negation of a property may be treated in a different way than the property itself. Further the format includes

- `trans`, where `trans`(R) means that R is transitive,
- `irr`, where `irr`(R) means that R is irreflexive,
- `compl`, where `compl`(R) means that R is complete,
- `nf`, where `nf`(x, R) means that x is a normal form with respect to R,
- `red`, where `red`(x, y, R) means that $(x, y) \in R$.

Finally, variables in the format can be assigned to newly created relations, like

- `union`, where `union`(R, S) represents the relation $R \cup S$,
- `comp`, where `comp`(R, S) represents the relation $R \cdot S$,
- `peak`, where `peak`(R, S) represents the relation $R^{-1} \cdot S$,
- `val`, where `val`(R, S) represents the relation $R \cdot S^{-1}$,
- `tc`, where `tc`(R) represents the transitive closure R^+ of R,
- `trc`, where `trc`(R) represents the transitive reflexive closure R^* of R.

Our tool `Carpa` reads a list of requirements in this format, and builds a formula for it representing these requirements in the way as described in this note. For every call of `sn` one new binary relation is created to represent S in Theorem 1, and to generate the corresponding requirements. For every call of `compl` two new binary relations are created to represent S and T in Theorem 3, and to generate the corresponding requirements. For every call of `trc` k new binary relations are created to generate the requirements as described in Theorem 2. Calls of `cr`, `wcr`, `un` and `wn` internally call `trc` and hence all cause the creation of at least k new binary relations.

In this format we described the requirements for the examples as they occur in this note, in fact these examples were found by applying our tool on the specifications written in this format.

The input always starts by three numbers: first the number $n = \#A$, next the number k as it is used for Theorem 2, and finally the number m of basic relations one is looking for, numbered from 1 to m. For instance, to obtain Example 4 we choose as input

```
6
3
3
x1=union(1,2)
compl(x1)
x1=union(1,3)
compl(x1)
x1=union(2,3)
compl(x1)
x1=union(1,x1)
ncr(x1)
```

Here the numbers state that we search for three relations on 6 elements. The next two lines state that the union of relations 1 and 2 should be complete. The next four lines do the same for the union of 1 and 3 and the union of 2 and 3. Finally, it is required that the union of all three relations is not confluent. The output of `Carpa` on this input reads

```
Relation 1:
(2,3)
(4,1)
(5,3)
Relation 2:
(1,5)
(2,4)
Relation 3:
(1,6)
(4,6)
(5,2)
```

indeed coinciding with Example 4.

References

[1] B. Dutertre and L. de Moura. Yices: An SMT solver. Available at `http://yices.csl.sri.com/`.

[2] 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.

[3] A. Stump, G. Kimmell, H. Zantema, and R. El Haj Omar. A rewriting view of simple typing. 2012. Submitted.

[4] V. van Oostrom. Confluence by decreasing diagrams. *Theoretical Computer Science*, 126(2):259–280, 1994.

[5] V. van Oostrom. Confluence by Decreasing Diagrams, Converted. In A. Voronkov, editor, *Proceedings of the 19th Conference on Rewriting Techniques and Applications (RTA)*, volume 5117 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2008.

[6] H. Zantema. Finding small counter examples for abstract rewriting properties. 2012. Submitted, available via `http://www.win.tue.nl/~hzantema/carpa.html`.

ACP: System Description for CoCo 2012

Takahito Aoto
RIEC
Tohoku University
aoto@nue.riec.tohoku.ac.jp

Yoshihito Toyama
RIEC
Tohoku University
toyama@nue.riec.tohoku.ac.jp

ACP is an automated confluence prover for term rewriting systems (TRSs) that has been developed in Toyama–Aoto group in RIEC, Tohoku University. Beside the most well-known criteria for proving confluence such as Knuth–Bendix criterion (1970) and Huet–Toyama–van Oostrom criterion (1997), ACP integrates many direct criteria for guaranteeing confluence of TRSs—see the website of ACP [1] for the list of implemented criteria. It incorporates divide–and–conquer criteria such as those for commutative (Toyama, 1988), layer-preserving (Ohlebusch, 1994) and persistent (Aoto & Toyama, 1997) combinations. For a TRS to which direct confluence criteria do not apply, the prover decomposes it into components using divide–and–conquer criteria, and tries to apply direct confluence criteria to each component. Then the prover combines these results to infer the (non-)confluence of the whole system.

ACP is written in Standard ML of New Jersey (SML/NJ) [7] and is provided as a heap image that can be loaded into SML/NJ runtime systems. It uses a SAT prover such as MiniSAT [5] and an SMT prover YICES [4] as external provers. It internally contains an automated (relative) termination prover for TRSs but external (relative) termination provers can be substituted optionally. The input TRS is specified in the (old) TPDB format [6]. Although the TPDB format allows to specify various rewriting strategies and rewriting modulo equations, proving confluence of TRSs under such varieties of rewriting is beyond the scope of the current version of ACP. Although a variety of criteria for proving confluence has been implemented, users can specify criteria to be used so that each criterion or any combination of them can be tested.

ACP participating in CoCo 2012 is based on version 0.30, which has been released in May, 2012. From this version, the source code became available (in a BSD License). The internal structure of the prover is kept simple and is mostly inherited from the version 0.11a, which has been described in [3]. Compared to the previous version, confluence criterion using reduction completion [2] has been newly incorporated. A relative termination prover has been also newly incorporated to support checking this criterion. Several levels of verbosity of the output have been introduced so that users can investigate details of the employed approximations for each criterion or can get only the final result of prover's attempt to use ACP as an external confluence prover.

References

[1] ACP (Automated Confluence Prover). http://www.nue.riec.tohoku.ac.jp/tools/acp/.

[2] T. Aoto and Y. Toyama. A reduction-preserving completion for proving confluence of non-terminating term rewriting systems. *LMCS*, 8(1:31):1–29, 2012.

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

[4] B. Dutertre and L. de Moura. The YICES SMT solver. http://yices.csl.sri.com/.

[5] N. Eén and N Sörensson. An extensible SAT-solver. In *Proc. of SAT 2003*, volume 2919 of *LNCS*, pages 502–518. Springer-Verlag, 2004. http://minisat.se/.

[6] C. Marché, A. Rubio, and H. Zantema. Termination Problem Data Base: format of input files. http://www.lri.fr/~marche/tpdb/format.html, 2005.

[7] Standard ML of New Jersey. http://www.sml.org/.

Certification of Confluence Proofs using CeTA*

René Thiemann, Institute of Computer Science, University of Innsbruck, Austria

Automatic provers have become popular in several areas like first-order theorem proving, SMT, ... Since these provers are complex pieces of software, they might contain errors which might lead to wrong answers, i.e., incorrect proofs. Therefore, certification of the generated proofs is of major importance, where soundness of the certifier itself might be proven in some trusted proof assistance like Coq [1] or Isabelle/HOL [5]. The tool CeTA is such a certifier [6]. Its soundness is proven in the corresponding IsaFoR-library (Isabelle Formalization of Rewriting) and CeTA can be used to check termination and non-termination proofs of term rewrite systems (TRSs). Starting from version 2.0, it is also possible to certify confluence and non-confluence proofs where the following techniques are currently supported in CeTA (version 2.5).

- Since CeTA's main domain are termination proofs, as a first method to decide confluence we integrated Newman's lemma in combination with the critical pair theorem [4]. Here, CeTA just rewrites both sides of a critical pair to arbitrary normal forms and then compares whether these normal forms coincide.
- For possibly non-terminating TRSs, we integrated the criterion of weak orthogonality to ensure confluence. The reason for not (yet) considering left-linear parallel-closed TRSs is the complexity of the soundness proof. For parallel-closed TRSs a complex measure on parallel forks is utilized [3], whereas for weak orthogonality structural induction suffices.
- To disprove confluence one can provide two forking derivations $s \to^* t_1$ and $s \to^* t_2$ in combination with a reason why t_1 and t_2 cannot be joined. Here, CeTA accepts the reason that t_1 and t_2 are distinct normal forms or alternatively, a test is performed using *tcap* [2, 7]: if $tcap(t_1\sigma)$ and $tcap(t_2\sigma)$ are not unifiable then a join is impossible (where σ is a substitution which replaces each variable x by some fresh constant c_x.)

For further details we refer to the certification problem format (CPF) and to the sources of IsaFoR and CeTA (`http://cl-informatik.uibk.ac.at/software/ceta/`). It remains as future and ongoing work to integrate existing and future confluence and non-confluence criteria.

References

[1] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development; Coq'Art: The Calculus of Inductive Constructions*. TCS Texts. Springer, 2004.

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

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

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

[5] T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[6] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *TPHOLs*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009.

[7] H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – A confluence tool. In *CADE*, volume 6803 of *LNAI*, pages 499–505, 2011.

*Supported by Austrian Science Fund (FWF): P22767

CoCo 2012 Participant: CSI*

Harald Zankl Bertram Felgenhauer Aart Middeldorp
Institute of Computer Science, University of Innsbruck, Austria

CSI is an automatic tool for (dis)proving confluence of first-order term rewrite systems (TRSs). Its name is derived from the Confluence of the rivers Sill and Inn in Innsbruck. The tool is open source and available from

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

CSI is based on the termination prover T$_{\text{T}}$T2. The main features and at the same time the major attractions of CSI are listed below. Several of these are described in more detail in [6].

- CSI is equipped with a strategy language, which allows to configure it flexibly. A web interface is also available.
- CSI implements the decreasing diagrams technique in a modular way, where different labelings can be combined lexicographically to obtain decreasingness.
- CSI supports decomposing TRSs into smaller TRSs based on ordered sorts (subtypes). This criterion is strictly stronger than a decomposition based on plain sorts. Apart from Knuth and Bendix' criterion this is currently the only method inside CSI which is applicable to non-left-linear systems.
- CSI features an efficient decision procedure for confluence of ground TRSs [2] that runs in cubic time in terms of the TRS size.
- Our non-confluence techniques employ methods from termination analysis, namely tcap, and tree automata techniques. For counterexamples to confluence we currently start with critical peaks. However, as shown in [3] this is not always sufficient, even for linear TRSs.
- CSI can produce proofs in cpf format that can be verified by certifiers like CeTA [5].

We conclude by mentioning a topic concerning future work. Currently, CSI is not very powerful for TRSs which are neither terminating nor left-linear. This is not due to the basic design but due to lack of techniques dedicated to this class of problems. Here ACP [1] and Saigawa [4] are clearly superior to our prover.

References

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

[2] B. Felgenhauer. Deciding confluence of ground term rewrite systems in cubic time. In *Proc. 23rd RTA*, LIPIcs, 2012. To appear.

[3] B. Felgenhauer. A proof order for decreasing diagrams. In *Proc. 1st IWC*, 2012. This volume.

[4] D. Klein and N. Hirokawa. Confluence of non-left-linear TRSs via relative termination. In *Proc. 18th LPAR*, volume 7180 of *LNCS (ARCoSS)*, pages 258–273, 2012.

[5] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proc. 22nd TPHOLs*, volume 5674 of *LNCS*, pages 452–468, 2009.

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

*Supported by the Austrian Science Fund (FWF) P22467.

Saigawa: A Confluence Tool*

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

Dominik Klein
JAIST, Japan
dominik.klein@jaist.ac.jp

1 System Description

Saigawa is a tool for automatically proving or disproving confluence of (ordinary) term rewrite systems (TRSs). The tool, written in OCaml, is freely available from

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

This system description is based on Saigawa version 1.3. The typical usage of the tool is: `saigawa <file>`. Here the input file is written in the standard WST format. The tool outputs `YES` if confluence of the input TRS is proved, `NO` if non-confluence is shown, and `MAYBE` if the tool does not reach any conclusion. The tool is based on the next three confluence criteria.

Theorem 1 ([1, Theorem 3]). *A left-linear TRS $\mathcal{R}$ is confluent if $\mathsf{CP}(\mathcal{R}) \subseteq \downarrow_{\mathcal{R}}$ and $\mathsf{CPS}'(\mathcal{R})/\mathcal{R}$ is terminating.*

Theorem 2 ([3, Theorem 2]). *Suppose $\mathcal{R}$ and $\mathcal{S}$ are strongly non-overlapping on each other and $\mathcal{S}$ is confluent. The TRS $\mathcal{R} \cup \mathcal{S}$ is confluent if and only if $\mathsf{CP}_{\mathcal{S}}(\mathcal{R}) \subseteq \downarrow_{\mathcal{R}\cup\mathcal{S}}$.*

Theorem 3 ([4]). *A TRS $\mathcal{R}$ is confluent if every critical peak is decreasing with respect to the rule labeling heuristic.*

Our tool uses T$_{\text{T}}$T$_2$ v1.07[1] to check (relative) termination. When termination of $\mathcal{R}$ is proved, the joinability $s \downarrow_{\mathcal{R}} t$ is tested by comparing normal forms of s and t. In the other cases we only test $s \to_{\mathcal{R}}^{m} \cdot {}_{\mathcal{R}}^{n}\!\leftarrow t$ for each $1 \leqslant m, n \leqslant 5$. Unjoinability is detected by testing whether $\mathsf{TCAP}_{\mathcal{R}}(s)$ and $\mathsf{TCAP}_{\mathcal{R}}(t)$ do not unify [5]. In order to apply Theorem 2 we need to appropriately split a TRS into $\mathcal{R}$ and $\mathcal{S}$. In our tool this is done by simple enumeration, and confluence of $\mathcal{S}$ is checked in a recursive manner. A suitable rule labeling is searched by using the SMT solver MiniSmt,[2] see [1, Section 4] for details of automation. We are planning to support Jouannaud and Kirchner's confluence criterion [2] in the next version.

References

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

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

[3] D. Klein and N. Hirokawa. Confluence of non-left-linear TRSs via relative termination. In *Proc. 18th LPAR*, volume 7180 of *LNCS*, pages 258–273, 2012.

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

[5] H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – a confluence tool. In *Proc. 23th CADE*, volume 6803 of *LNAI*, pages 499–505, 2011.

*This work is supported by the Grant-in-Aids for Young Scientists (B) 22700009 and Scientific Research (B) 23300005 of the Japan Society for the Promotion of Science.

[1] http://colo6-c703.uibk.ac.at/ttt2/

[2] http://cl-informatik.uibk.ac.at/software/minismt/

Author Index