IWC 2017

6th International Workshop on Confluence

Proceedings

Editors: Beniamino Accattoli & Bertram Felgenhauer

September 8th, 2017, Oxford, United Kingdom

Preface

This report contains the proceedings of the 6th International Workshop on Confluence (IWC 2017), which was held in Oxford, United Kingdom on September 8th, 2017. The workshop was part of the 2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017). Previous IWC workshops were held in Nagoya (2012), Eindhoven (2013), Vienna (2014), Berlin (2015), and Obergurgl (2016).

Confluence provides a general notion of determinism and has been conceived as one of the central properties of rewriting. Confluence relates to many topics of rewriting (completion, modularity, termination, commutation, etc.) and had been investigated in many formalisms of rewriting such as first-order rewriting, lambda-calculi, higher-order rewriting, constrained rewriting, conditional rewriting, etc. Recently there is a renewed interest in confluence research, resulting in new techniques, tool support, certification as well as new applications. The workshop promotes and stimulates research and collaboration on confluence and related properties. In addition to original contributions, the workshop solicited short versions of recently published articles and papers submitted elsewhere.

IWC 2017 received 10 submissions. Almost all submissions were reviewed by 3 program committee members. After deliberations, the program committee decided to accept all submissions, which are contained in this report. Apart from these contributed talks, the workshop had two invited talks. One invited talk was given by Stefan Kahrs, on Consistency and Invariants. The second invited talk by Patrick Bahr was about Böhm Reduction for Terms and Term Graphs. Their abstracts are also included in the report. Moreover, the 6th Confluence Competition (CoCo 2017) was held during the workshop and the results are available at `http://coco.nue.riec.tohoku.ac.jp/2017/`.

Several people helped to make IWC 2017 a success. We are grateful to the members of the program committee for their work. We also thank the members of the FSCD organizing committee for hosting IWC 2017 in Oxford.

August 2017

Beniamino Accattoli
Bertram Felgenhauer

Organization

IWC 2017 was part of the 2nd International Conference on Formal Structures in Computation and Deduction (FSCD 2017), which was organized by the Department of Computer Science at the University of Oxford.

Program Committee Chairs

Beniamino Accatolli	INRIA
Bertram Felgenhauer	University of Innsbruck

Program Committee

Beniamino Accatolli	INRIA
Jörg Endrullis	Vrije Universiteit Amsterdam
Bertram Felgenhauer	University of Innsbruck
Ken-Etsu Fujita	Gumma University
Philippe Malbos	Université Claude Bernard Lyon 1
Jakob Grue Simonsen	University of Copenhagen

Table of Contents

Consistency and Invariants — Extended Abstract

Stefan Kahrs

School of Computing
University of Kent
Canterbury, United Kingdom
S.M.Kahrs@kent.ac.uk

Abstract

We are looking at the specific problem of proving consistency of TRSs and the more general problem of proving program invariants. On the specific side we show that almost non-ω-overlapping constructor TRSs are consistent (and have unique normal forms), but the proof technique to get there shows us ways how to tackle the much more general problem of proving program invariants. To prove invariants we are using *proof graphs*, union/find-structures that carry proof information. As index set for these structures we use finite Σ-coalgebras — this is sufficient, because any equational proof is a finite object.

1 Introduction

Note: this is based on a paper that appeared in FSCD 2016 [7].

For over 40 years [10] it has been known that TRSs that are left-linear and non-overlapping are confluent, and for over 30 years [6] that non-overlapping on its own may not even give us unique normal forms:

Example 1. *By Huet [6]:* $\{F(x,x) \to A, F(x,G(x)) \to B, C \to G(C)\}$. *The term* $F(C,C)$ *possesses two distinct normal forms,* A *and* B.

Moreover, if we replace the right-hand side of the first rule with x then the TRS is even inconsistent. The first two rules overlap syntactically, in a certain sense: if we can instantiate x with the infinite term $G(G(\cdots))$ then the first two rules would both be redexes, i.e. the rules ω-overlap. This creates the question: *do non-ω-overlapping TRSs have unique normal forms*? This was first conjectured in the 1980s by Ogawa [9]. Not only can UN be replaced by CON, we can simplify the problem from TRSs to constructor TRSs, because there is a translation between such systems [7] which preserves and reflects consistency, and which (modulo some minor issues) also preserves the non-ω-overlapping property.

Properties such as confluence, uniqueness of normal forms, and consistency are examples of program invariants, properties that are implied by $t =_R u$. Or rather: a confluence proof shows that the joinability relation $\downarrow$ is a program invariant: if $t =_R u$ then $t \downarrow u$. Of all program invariants, consistency is the bottom line: if $t =_R u$ is vacuously true then we cannot deduce anything from it. Therefore, whenever we prove a non-vacuous program invariant, we should get a proof of consistency for free.

In general, whether a property is an invariant is undecidable for TRSs, so we are looking for classes of TRSs for which the problem is easier, and sometimes for over-approximations of our invariants. What we typically have as an *invariant candidate* is an inductively defined relation which is in general not a congruence relation, but might be for well-behaved TRSs. Moreover, the missing ingredient is often transitivity, e.g. if $\downarrow$ is transitive then it is also a congruence relation.

How does one show that a certain inductively defined invariant candidate $\triangle = \mu x.F(x)$ is indeed an invariant, i.e. in particular that it is transitive? Instead of considering whether

B. Accatoli and B. Felgenhauer (eds.); IWC 2017, pp. 1–6

$t =_R u$ or $t \triangle u$ do or do not hold *in general* (typically undecidable properties), we consider whether we can prove them if we are only allowed to use in the proofs terms from some finite set A — more specifically, we want that set to be closed under subterms, which means that the set is a coalgebra of the signature functor. For this finitist argument to work, we need that F is continuous, that we never need an infinite part of the relation $\triangle$ to deduce $t \triangle u$. The condition "only terms from A are allowed" can typically be achieved by defining $\triangle_A = \mu x.id_A \cdot F(x) \cdot id_A$, where id_A is the identity on that set A.

If that relation $\triangle_A$ is always transitive we must have that $\triangle$ itself is always transitive. Even if that were not the case it would suffice if we could find, for any finite coalgebra A, a coalgebra $B \supset A$, and an equivalence relation $=_B$ such that $\triangle_A \subseteq =_B \subseteq \triangle_B$.

2 Proof Graphs

One way of showing that a (finite) relation $\triangle$ is transitive is to build a complete finite representation of it that necessarily implies the transitivity of the relation. For this purpose we are using union/find-structures [5, 12]. Union/find-structures are generally used to represent finite equivalence relations in an efficient way; in this context their efficiency is not a particular concern though it is related to the relative simplicity of the manipulation of the equivalence. A special case of proof graph to support confluence proofs had been used by Sakai et.al.[11].

What we are looking for are union/find-structures ϱ whose associate equivalence relations $=_\varrho$ are necessarily subrelations of our relation $\triangle$. For this to get off the ground we need at least that $\triangle$ is reflexive and symmetric.

Definition 1. *An* equivalence candidate *is a reflexive and symmetric relation.*

Secondly, to achieve the goal $=_\varrho \subseteq \triangle$ we need that the edges used in our structure have sufficiently "nice" properties. Ideally, this means that they are "prefixes":

Definition 2. *A* prefix *of an equivalence candidate $\triangle$ is a relation $\lhd$ such that $\lhd \cdot \triangle \subseteq \triangle$.*

It is easy to see that prefixes of $\triangle$ have various nice algebraic properties: they are closed under arbitrary union, composition, and subrelations. A union/find-structure ϱ is then a proof graph for $\triangle$ if its edge relation $\xrightarrow{\varrho}$ is a prefix of $\triangle$. This suffices to imply $=_\varrho \subseteq \triangle$.

Notice that if $\triangle$ is transitive then it is itself its own prefix. However, (i) we do not know that to begin with but we may know that about a relation $\lhd$; (ii) if we constrain our edges to use a particular kind of prefix then we can also exploit that knowledge.

The idea behind the notion of proof graph is that our equivalence candidates are inductively defined relations that trivially have certain relations as prefixes, as part of their definition.

Sometimes, we need to prove the transitivity of a candidate $\triangle_A$ indirectly, through a proof graph for $\triangle_B$. This works as follows:

Definition 3. *Let (A, α) and (B, β) be proof graphs for candidates $\triangle_A$ and $\triangle_B$, respectively. Then a function $f : A \to B$ is a* proof graph morphism *iff we have the following properties:*

- $\forall x, y \in A.\, x \xrightarrow{\alpha} y \Rightarrow f(x) \xrightarrow{\beta} f(y)$
- $\forall x, y \in A.\, f(x) \triangle_B f(y) \Rightarrow x \triangle_A y$

Clearly, proof graph morphisms can be composed and thus form a category. Moreover, each proof graph morphism f induces an equivalence on its domain A: $x =_f y \iff f(x) =_\beta f(y)$, and by the second condition this equivalence must be a subrelation of $\triangle_A$.

Generally, we have two typical ways of creating proof graph morphisms: (i) by adding edges to the graph, without changing the candidate or index set, and (ii) by embedding A into a larger set B, preserving all edges, but constraining the relation $\triangle_B$ suitably. For example, we may want to do this for a confluence proof by providing in B missing common reducts whilst restricting root rewrite steps to reside within A.

One thing we can do with proof graphs is add prefix edges to a normal forms. If we want to limit ourselves to edges of a particular prefix $\lhd$ we can always extend a given proof graph (via proof graph morphisms), so that the resulting proof graph ω has the UN property for proof graphs:

Definition 4. *A proof graph ω has the* UN property for prefix $\lhd$ *iff*

$$\forall t, s.\ (\neg(\exists u.\ t \xrightarrow{\omega} u)) \Rightarrow t \lhd^* s \Rightarrow t =_\omega s$$

.

The reason this is called a "UN property", because if two normal forms ("roots" in union/find-structure terminology) of the proof graph have a common $\lhd$ reduct then they must be the same.

Lemma 1. *Any proof graph has an extension that has the UN property for any prefix of its equivalence candidate.*

Compared to considering the set of all terms we gain additional induction principles when organising a set of terms as index set for a proof graph ω — as we limit ourselves to finitely many terms, finitely many equivalence classes, and in particular: a terminating edge relation $\xrightarrow{\omega}$.

3 Term-Coalgebras, and relations on them

Relations on terms can more generally be viewed as relations on or between Σ-coalgebras. This can be useful to stratify the reasoning on terms, one finite Σ-coalgebra at a time.

In order to consider coalgebras of signatures Σ we would have to view signatures as functors on the category Set. However, we only need here the following special instance of this concept:

Definition 5. *Given a signature Σ, a* term-coalgebra *is a set $A \subseteq Ter^\infty(\Sigma, \emptyset)$ which is closed under subterms. It is called* finite *if it is a finite set, and* strongly finite *if in addition $A \subseteq Ter(\Sigma, \emptyset)$.*

More generally, Σ-coalgebras A would be characterized by a function $\upsilon : A \to \Sigma(A)$ which maps a node to a structure containing its root function symbol and the list of its subnodes. We also allow for variables in term-coalgebras by "freezing" them, i.e. when considered as a member of a term-coalgebra a variable is a nullary constructor. For heterogeneous relations between term-coalgebras we must therefore have that the variable set X is the same, so that they are coalgebras of the same functor.

One ingredient to define relations between or on term-coalgebras for a signature Σ we use the following notation: if $R \subseteq A \times B$, where A and B are term-coalgebras A and B then $\widetilde{R} \subseteq A \times B$ is defined as follows:

$$\begin{aligned}\forall t \in A.\, \forall u \in B.\, t\ \widetilde{R}\ u \iff\ & \exists F \in \Sigma.\, \exists a_1, \ldots, a_n \in A.\, \exists b_1, \ldots, b_n \in B.\\ & t = F(a_1, \ldots, a_n) \wedge u = F(b_1, \ldots, b_n) \wedge \forall i.\, a_i\ R\ b_i\end{aligned}$$

This concept was first used in [3, 4] (though with added reflexivity). For constructur signatures, we use the notations $\overline{R}$ and $\widehat{R}$ to mean $\widetilde{R}$ for the subsignatures Σ_d and Σ_c, respectively. In particular, $t\ \widehat{id}\ t$ iff the root symbol of t is a constructor, and so $\widehat{R} \cdot \overline{S} = \emptyset$. We still use $\widetilde{R}$ for constructor signatures, to refer to the combined signature; hence $\widetilde{R} = \overline{R} \cup \widehat{R}$.

If id_V is the coreflexive identity on variables then we can express consistency of a relation R relation-algebraically as $id_V \cdot R \cdot id_V \subseteq id_V$. However, we already have consistency issues when a relation R relates any terms topped with distinct constructors. Relations that do not do that we call "constructor-consistent": $\widehat{id} \cdot R \cdot \widehat{id} \subseteq \widehat{1}$, where "1" is top element of the lattice of relations. To reason about pattern matching we need something even stronger than that:

Definition 6. *A relation R between term-coalgebras is called* constructor-compatible *iff*

$$\widehat{id} \cdot R \cdot \widehat{id} \subseteq \widehat{R}.$$

Constructor-compatible relations are preserved by arbitrary union; consequently, relations defined as $\mu x.\, f(x)$ are constructor-compatible whenever the function f preserves this property. The same applies to consistent or constructor-consistent relations.

For "well-behaved" Constructor TRSs we would expect these forms of consistency to be properties of the standard rewrite congruence $=_R$. However, constructor-compatibility can sometimes be lacking in strange ways:

Example 2. $F(x,x) \to x, F(y, G(G(y))) \to G(G(G(y))), A \to G(G(A))$ *These rules are almost non-ω-overlapping.*

In Example 2, we have $G(G(A)) =_R G(G(G(A)))$ but we do not have $G(A) =_R G(G(A))$, i.e. $=_R$ is not constructor-compatible. The issue goes away (globally) if we request that for all constructors there are matching projection operations, e.g. here we would need a symbol π_G with rewrite rule $\pi_G(G(x)) \to x$; locally (at finite term-coalgebras) the issue would persist unless we ruled out term-coalgebras that contained $G(t)$ without containing $\pi_G(G(t))$. Alternatively, we can use an equivalence which over-approximates $=_R$. Any invariant of such an over-approximation must be an invariant of the original relation.

4 Redex Behaviour

Definition 7. *A relation R between term-coalgebras is called* Σ-closed *iff* $\widetilde{R} \subseteq R$.

Note: this is standard terminology taken from [1], except that we generalise it to coalgebras. When dealing with almost non-ω-overlapping constructor-rewriting, we sometimes require a stronger form of Σ-closure, because trivial ω-overlaps can cause anomalies, as we have seen in Example 2.

Definition 8. *An equivalence relation $=_\varrho$ on a term-coalgebra A is called* strongly Σ_c-closed *iff* $\nu x.\, =_\varrho \cdot \widehat{x} \cup \widehat{x} \cdot =_\varrho \subseteq =_\varrho$.

Notice that the construction in the definition uses a largest fixpoint, which means that even congruence relations may not have this property. For instance, let $=_\varrho$ be the strong Σ_c-closure of $=_R$ of Example 2, then we now do have $G(A) =_\varrho G(G(A))$ and $A =_\varrho G(A)$.

Proposition 1. *Let $a \to b$ and $c \to d$ be two rewrite rules (of some constructor TRS) with only trivial ω-overlaps. Let $=_S$ be a constructor-compatible and Σ-closed equivalence. Then $\sigma(a) \equiv_{\overline{S}} \theta(c)$ implies $\sigma(b) =_S \theta(d)$.*

The reason this is true is that (i) every equation derived via the ω-unification algorithm still holds in $=_S$, (ii) Σ-closed equivalences "are" algebras, so that we can "interpret" eventually the anti-unifiers of b and d in $=_S$-equivalent environments, giving $=_S$-equivalent results.

5 An invariant

Given a TRS with signature Σ, and a term-coalgebra A, the relation $\Downarrow_A$ is a relation on A defined as follows:

$$\Downarrow_A \doteq \mu x.\, \nu y.\, (id_A \cdot (\overline{x} \cup \widehat{y}) \cdot id_A)^* \cdot (\xrightarrow{\epsilon}\cdot\, x \cup id \cup x \cdot\xleftarrow{\epsilon}) \cdot (id_A \cdot (\overline{x} \cup \widehat{y}) \cdot id_A)^*$$

The reason for the largest fixpoint in the definition are almost-non-ω-overlapping systems such as Example 2, i.e. we intend to use $\Downarrow_A$ as an invariant for these systems. The relation $\Downarrow$ bears some similarity to joinability, but has one stronger feature: we have that $\widetilde{\Downarrow}$ is a prefix of $\Downarrow$. Unlike joinability, $\Downarrow$ is not constructor-compatible (in general), but it is constructor-consistent. One can define similar invariants [7] that are constructor-compatible too, but these fit less well with our general notion of proof graph, in particular we would for those relations not have the corresponding property that $\widehat{\Downarrow_A}$ is a prefix of $\Downarrow_A$.

6 Building a Universal Proof Graph

To show that our proposed invariant relation $\Downarrow_A$ is indeed an invariant for almost non=ω-overlapping constructor TRSs we can show instead that it is transitive — because the universal properties of $=_R$ would imply the rest. For this purpose we need to build a *universal proof graph* for $\Downarrow_A$. This is a proof graph ϱ on A such that $=_\varrho$ and $\Downarrow_A$ coincide.

In general, this kind of construction might be indirect via a proof graph morphism f, i.e. through a proof graph for $\Downarrow_B$ we would get that $=_f$ coincides with $\Downarrow_A$, not the proof graph equivalences themselves. For consistency proofs we can typically stick to one coalgebra and one invariant.

Definition 9. *We build a proof graph ϱ for $\Downarrow_A$ as follows:*

- *Form PG α whose equivalence contains $id_A \cdot\xrightarrow{\epsilon}\cdot\, id_A$.*
- *Form extension β of this PG which has UN property w.r.t. prefix $\overline{\Downarrow_A}$.*
- *Form PG ϱ whose equivalence is the strong Σ_c-closure of $=_\beta$.*

Notice that we can form α whenever a TRS is non-overlapping (even when it is ω-overlapping). We can form β by Lemma 1; the knowledge of what the relation $\overline{\Downarrow_A}$ actually is, is a separate issue — it would typically require a separate data structure. Because the equivalence relation $\overline{\Downarrow_A}^*$ is a prefix too one can use a separate union/find structure for this purpose. Finally, the strong Σ_c-closure can always be performed on proof graphs of this relation, regardless of TRS.

For any inductively defined relation $\mu x.F(x)$ to prove that it coincides with $=_\varrho$ means to show that the inclusion $F(=_\varrho) \subseteq =_\varrho$ holds, i.e. $\forall t, u \in A.\, t\ F(=_\varrho\ u \Rightarrow t =_\varrho u$. If the equivalence $=_\varrho$ is derived from a proof graph, then we can show the property by induction on $\xrightarrow{\varrho}$. First, we define a well-founded ordering on pairs of terms from A:

$$(t, u) > (t', u') \iff t \xrightarrow{\varrho}{}^{+} t' \wedge u \xrightarrow{\varrho}{}^{*} u' \vee t \xrightarrow{\varrho}{}^{*} t' \wedge u \xrightarrow{\varrho}{}^{+} u'$$

Then we prove our goal by induction on this ordering, i.e. we are showing:

$$\forall t, u \in A.\, (\forall t', u' \in A.\, (t, u) > (t', u') \wedge t' \Downarrow_A u' \Rightarrow t' =_\varrho u') \Rightarrow (t\ F(=_\varrho)\ u \Rightarrow t =_\varrho u)$$

Proposition 2. *The proof graph ϱ is universal for $\Downarrow_A$ whenever the TRS is almost non-ω-overlapping.*

Theorem 1. *Almost non-ω-overlapping Constructor TRSs have a constructor-consistent equational theory.*

References

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

[2] Hendrik P. Barendregt. *The Lambda-Calculus, its Syntax and Semantics.* North-Holland, Amsterdam, 1984.

[3] Jörg Endrullis, Helle Hvid Hansen, Dimitri Hendriks, Andrew Polonsky, and Alexandra Silva. A coinductive treatment of infinitary rewriting. In *Workshop on Infinitary Rewriting*, page 8, 2013.

[4] Jörg Endrullis, Dimitri Hendriks, Helle Hvid Hansen, Andrew Polonsky, and Alexandra Silva. A coinductive framework for infinitary rewriting and equational reasoning. In *Rewriting Techniques and Applications*, 2015.

[5] Bernard A. Galler and Michael J. Fisher. An improved equivalence algorithm. *Commun. ACM*, 7(5):301–303, May 1964.

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

[7] Stefan Kahrs and Connor Smith. Non-Omega-Overlapping TRSs are UN. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, volume 52 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[8] Jan Willem Klop. *Combinatory Reduction Systems.* PhD thesis, Centrum voor Wiskunde en Informatica, 1980.

[9] Mizuhito Ogawa and Satoshi Ono. On the uniquely converging property of nonlinear term rewriting systems. Technical Report IEICE COMP89-7, NTT Software Laboratories, 1989.

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

[11] Masahiko Sakai, Michio Oyamaguchi, and Mizuhito Ogawa. Non-e-overlapping, weakly shallow, and non-collapsing trss are confluent. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2015.

[12] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, April 1975.

Böhm Reduction for Terms and Term Graphs

Patrick Bahr

IT University of Copenhagen, Denmark
paba@itu.dk

Abstract

Infinitary rewriting endows a rewriting system with a mode of convergence that assigns an outcome to each infinite reduction that is – in some sense – well-formed, for example, an infinite reduction that produces the infinite list of all natural numbers. Unfortunately, infinitary rewriting breaks well-known confluence results for lambda calculi and orthogonal term rewriting systems. In order to recover the confluence property in a meaningful way, Kennaway et al. extended the ordinary reduction relation so that *'meaningless terms'* can be contracted to a fresh constant $\bot$. They showed that this extended reduction relation – called Böhm reduction – enjoys the confluence property for infinitary lambda calculi and infinitary orthogonal term rewriting.

In this talk I give an overview of Böhm reduction and confluence in infinitary term rewriting. The underlying mode of convergence of Böhm reduction is based on metric spaces. As an alternative to this approach, I present a mode of convergence – based on partially ordered sets – that enjoys the confluence property for orthogonal term rewriting systems directly, i.e. without the need to explicitly contract meaningless terms. Finally, I sketch how a corresponding infinitary rewriting theory can be developed for term graphs.

B. Accatoli and B. Felgenhauer (eds.); IWC 2017, pp. 7–7

Certified Non-Confluence with ConCon 1.5*

Thomas Sternagel and Christian Sternagel

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

We present three methods to check CTRSs for non-confluence: (1) an ad hoc method for 4-CTRSs, (2) a specialized method for unconditional critical pairs, and finally, (3) a method that employs conditional narrowing to find non-confluence witnesses. We shortly describe our implementation of these methods in ConCon [8], then look into their certification with CeTA [11], and finally conclude with experiments on the confluence problems database (Cops).[1]

1 Preliminaries

We assume familiarity with the basic notions of (conditional) term rewriting [1, 4], but shortly recapitulate terminology and notation that we use in the remainder. Given an arbitrary binary relation $\to$, we write $\leftarrow$, $\to^+$, $\to^*$ for *inverse*, *transitive closure*, and *reflexive transitive closure*, respectively. We use $\mathcal{V}(\cdot)$ to denote the set of variables occurring in a given syntactic object, like a term, a pair of terms, a list of terms, etc. The set of terms $\mathcal{T}(\mathcal{F}, \mathcal{V})$ over a given signature of function symbols $\mathcal{F}$ and set of variables $\mathcal{V}$ is defined inductively: $x \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ for all variables $x \in \mathcal{V}$, and for every n-ary function symbol $f \in \mathcal{F}$ and terms $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ also $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. We say that terms s and t *unify* if $s\sigma = t\sigma$ for some substitution σ. The topmost part of a term that does not change under rewriting (sometimes called its "cap") can be approximated for example by the tcap function [2]. Informally, $\mathsf{tcap}(x)$ for a variable x results in a fresh variable, while $\mathsf{tcap}(t)$ for a non-variable term $t = f(t_1, \ldots, t_n)$ is obtained by recursively computing $u = f(\mathsf{tcap}(t_1), \ldots, \mathsf{tcap}(t_n))$ and then asserting $\mathsf{tcap}(t) = u$ in case u does not unify with any left-hand side of rules in the TRS $\mathcal{R}$, and a fresh variable, otherwise. We call a bijective variable substitution $\pi : \mathcal{V} \to \mathcal{V}$ a *(variable) renaming*. A *CTRS* $\mathcal{R}$ is a set of conditional rewrite rules of the shape $\ell \to r \Leftarrow c$ where ℓ and r are terms and c is a (possibly empty) sequence of pairs of terms $s_1 \approx t_1, \ldots, s_k \approx t_k$. For all rules in $\mathcal{R}$ we have that $\ell \notin \mathcal{V}$. If additionally $\mathcal{V}(r) \subseteq \mathcal{V}(\ell, c)$ for all rules we call $\mathcal{R}$ a 3-CTRS otherwise a 4-CTRS. We restrict our attention to *oriented* CTRSs where conditions are interpreted as reachability requirements. The rewrite relation induced by an oriented CTRS $\mathcal{R}$ is structured into *levels*. For each level i, a TRS $\mathcal{R}_i$ is defined recursively as follows: $\mathcal{R}_0 = \varnothing$, and $\mathcal{R}_{i+1} = \{\ell\sigma \to r\sigma \mid \ell \to r \Leftarrow c \in \mathcal{R}, \forall s \approx t \in c.\, s\sigma \to^*_{\mathcal{R}_i} t\sigma\}$. The rewrite relation of $\mathcal{R}$ is defined as $\to_{\mathcal{R}} = \bigcup_{i \geqslant 0} \to_{\mathcal{R}_i}$. By dropping all conditions from a CTRS $\mathcal{R}$ we obtain its *underlying TRS*, denoted $\mathcal{R}_{\mathsf{u}}$. Note that $\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}_{\mathsf{u}}}$. We sometimes label rules like $\rho\colon \ell \to r \Leftarrow c$. Two variable-disjoint variants of rules $\ell_1 \to r_1 \Leftarrow c_1$ and $\ell_2 \to r_2 \Leftarrow c_2$ in $\mathcal{R}$ such that $\ell_1|_p \notin \mathcal{V}$ and $\ell_1|_p\mu = \ell_2\mu$ with most general unifier (mgu) μ, constitute a *conditional overlap*. A conditional overlap that does not result from overlapping two variants of the same rule at the root, gives rise to a *conditional critical pair* (CCP) $\ell_1[r_2]_p\mu \approx r_1\mu \Leftarrow c_1\mu, c_2\mu$. A CCP $u \approx v \Leftarrow c$ is *joinable* if $u\sigma \to^*_{\mathcal{R}} \cdot \,^*_{\mathcal{R}}\!\leftarrow v\sigma$ for all substitutions σ such that $s\sigma \to^*_{\mathcal{R}} t\sigma$ for all $s \approx t \in c$. Moreover, a CCP $u \approx v \Leftarrow c$ is *infeasible* if there is no substitutions σ such that $s\sigma \to^*_{\mathcal{R}} t\sigma$ for all $s \approx t \in c$.

*This work is supported by FWF (Austrian Science Fund) project P27502.

[1] http://cops.uibk.ac.at/?q=ctrs+oriented

2 Finding Witnesses for Non-Confluence of CTRSs

To prove non-confluence of a CTRS we have to find a witness, that is, two diverging rewrite sequences starting at the same term whose end points are not joinable.

The first criterion only works for CTRSs that contain at least one unconditional rule of type 4, that is, with extra-variables in the right-hand side.

Lemma 1. *Given a 4-CTRS $\mathcal{R}$ and an unconditional rule $\rho\colon \ell \to r$ in $\mathcal{R}$ where $\mathcal{V}(r) \nsubseteq \mathcal{V}(\ell)$ and r is a normal form with respect to $\mathcal{R}_{\mathsf{u}}$ then $\mathcal{R}$ is non-confluent.*

Proof. Since $\mathcal{V}(r) \nsubseteq \mathcal{V}(\ell)$ we can always find two renamings μ_1 and μ_2 restricted to $\mathcal{V}(r) \setminus \mathcal{V}(\ell)$ such that $r\mu_1 \,{}_{\rho}\!\leftarrow \ell\mu_1 = \ell\mu_2 \to_{\rho} r\mu_2$ and $r\mu_1 \neq r\mu_2$. As r is a normal form with respect to $\mathcal{R}_{\mathsf{u}}$ also $r\mu_1$ and $r\mu_2$ are (different) normal forms with respect to $\mathcal{R}_{\mathsf{u}}$ (and hence also with respect to $\mathcal{R}$). Because we found a non-joinable peak $\mathcal{R}$ is non-confluent. □

Example 2. Consider the second (unconditional) rule of the 4-CTRS $\mathcal{R}_{320} = \{\mathsf{e} \to \mathsf{f}(x) \Leftarrow \mathsf{l} \approx \mathsf{d},\ \mathsf{A} \to \mathsf{h}(x,x)\}$ from Cops. Its right-hand side $\mathsf{h}(x,x)$ is a normal form with respect to the underlying TRS of $\mathcal{R}_{320}$ and the only variable occurring in it does not appear in its left-hand side A. So by Lemma 1 $\mathcal{R}_{320}$ is non-confluent.

A natural candidate for diverging situations are the critical peaks of a CTRS. We will base our next criterion on the analysis of *unconditional* critical pairs (CPs) of CTRSs. This restriction is necessary to guarantee the existence of the actual peak. If we would also allow conditional CPs, we first would have to check for infeasibility, since infeasibility is undecidable in general these checks are potentially very costly (see for example [9]).

Lemma 3. *Given a CTRS $\mathcal{R}$ and an unconditional CP $s \approx t$ of it. If s and t are not joinable with respect to $\mathcal{R}_{\mathsf{u}}$ then $\mathcal{R}$ is non-confluent.*

Proof. The CP $s \approx t$ originates from a critical overlap between two unconditional rules $\rho_1\colon \ell_1 \to r_1$ and $\rho_2\colon \ell_2 \to r_2$ for some mgu μ of $\ell_1|_p$ and ℓ_2 such that $s = \ell_1\mu[r_2\mu]_p \leftarrow \ell_1\mu[\ell_2\mu]_p \to r_1\mu = t$. Since s and t are not joinable with respect to $\mathcal{R}_{\mathsf{u}}$ they are of course also not joinable with respect to $\mathcal{R}$ and we have found a non-joinable peak. So $\mathcal{R}$ is non-confluent. □

Example 4. Consider the 3-CTRS $\mathcal{R}_{271} = \{\mathsf{p}(\mathsf{q}(x)) \to \mathsf{p}(\mathsf{r}(x)),\ \mathsf{q}(\mathsf{h}(x)) \to \mathsf{r}(x),\ \mathsf{r}(x) \to \mathsf{r}(\mathsf{h}(x)) \Leftarrow \mathsf{s}(x) \approx \mathsf{0},\ \mathsf{s}(x) \to \mathsf{1}\}$ from Cops. First of all we can immediately drop the third rule because we can never satisfy its condition and so it does not influence the rewrite relation of $\mathcal{R}_{271}$. This results in the TRS $\mathcal{R}'_{271}$. Now the left- and right-hand sides of the unconditional CP $\mathsf{p}(\mathsf{r}(z)) \approx \mathsf{p}(\mathsf{r}(\mathsf{h}(z)))$ are not joinable because they are two different normal forms with respect to the underlying TRS of $\mathcal{R}'_{271}$. Hence $\mathcal{R}_{271}$ is not confluent by Lemma 3.

While the above lemmas are easy to check and we have fast methods to do so they are also rather ad hoc. A more general but potentially very expensive way to search for non-joinable forks is to use conditional narrowing [3].

Definition 5 (Conditional narrowing)**.** Given a CTRS $\mathcal{R}$ we say that s *(conditionally) narrows* to t, written $s \rightsquigarrow_{\sigma} t$ if there is a variant of a rule $\rho\colon \ell \to r \Leftarrow c \in \mathcal{R}$, such that $\mathcal{V}(s) \cap \mathcal{V}(\rho) = \varnothing$ and $u \rightsquigarrow_{\sigma}^{*} v$ for all $u \approx v \in c$, a position $p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(s)$, a unifier[2] σ of $s|_p$ and ℓ, and $t = s[r]_p\sigma$. For a narrowing sequence $s_1 \rightsquigarrow_{\sigma_1} s_2 \rightsquigarrow_{\sigma_2} \cdots \rightsquigarrow_{\sigma_{n-1}} s_n$ of length n we write $s_1 \rightsquigarrow_{\sigma}^{n} s_n$ where $\sigma = \sigma_1\sigma_2\cdots\sigma_{n-1}$. If we are not interested in the length we also write $s \rightsquigarrow_{\sigma}^{*} t$.

The following property of narrowing carries over from the unconditional case:

[2] In our implementation we start from an mgu of $s|_p$ and ℓ and extend it while trying to satisfy the conditions.

Property 6. *If $s \rightsquigarrow_\sigma t$ then $s\sigma \to t\sigma$ with the same rule that was employed in the narrowing step. Moreover, if $s_1 \rightsquigarrow_{\sigma_1} s_2 \rightsquigarrow_{\sigma_2} \cdots \rightsquigarrow_{\sigma_{n-1}} s_n$ then $s_1\sigma_1\sigma_2\cdots\sigma_{n-1} \to s_2\sigma_2\cdots\sigma_{n-1} \to \cdots \to s_n$. Again employing the same rule for each rewrite step as in the corresponding narrowing step.*

Using conditional narrowing we can now formulate a more general non-confluence criterion.

Lemma 7. *Given a CTRS $\mathcal{R}$, if we can find two narrowing sequences $u \rightsquigarrow^*_\sigma s$ and $v \rightsquigarrow^*_\tau t$ such that $u\sigma\mu = v\tau\mu$ for some mgu μ and $s\sigma\mu$ and $t\tau\mu$ are not $\mathcal{R}_\mathsf{u}$-joinable then $\mathcal{R}$ is non-confluent.*

Proof. Employing Property 6 we immediately get the two rewriting sequences $u\sigma \to^*_\mathcal{R} s\sigma$ and $v\tau \to^*_\mathcal{R} t\tau$. Since rewriting is closed under substitutions we have $s\sigma\mu \;{}^*_\mathcal{R}\!\leftarrow u\sigma\mu = v\tau\mu \to^*_\mathcal{R} t\tau\mu$. As the two endpoints of these forking sequences $s\sigma\mu$ and $t\tau\mu$ are not joinable by $\mathcal{R}_\mathsf{u}$ they are certainly also not joinable by $\mathcal{R}$. This establishes non-confluence of the CTRS $\mathcal{R}$. □

Example 8. Consider the 3-CTRS $\mathcal{R}_{262} = \{0{+}y \to y,\ \mathsf{s}(x){+}y \to x{+}\mathsf{s}(y),\ \mathsf{f}(x,y) \to z \Leftarrow x{+}y \approx z + z'\}$ from Cops. Starting from a variant of the left-hand side of the third rule $u = \mathsf{f}(x', y')$ we have a narrowing sequence $\mathsf{f}(x', y') \rightsquigarrow_\sigma x_1$ using the variant $\mathsf{f}(x_1, x_2) \to x_3 \Leftarrow x_1 + x_2 \approx x_3 + x_4$ of the third rule and the substitution $\sigma = \{x' \mapsto x_1, x_3 \mapsto x_1, x_4 \mapsto x_2\}$. We also have another narrowing sequence $\mathsf{f}(x', y') \rightsquigarrow_\tau x_3$ using the same variant of rule three and substitution $\tau = \{x \mapsto x_3 + x_4, x' \mapsto \mathsf{0}, y' \mapsto x_3 + x_4, x_1 \mapsto \mathsf{0}, x_2 \mapsto x_3 + x_4\}$ where for the condition $x_1 + x_2 \approx x_3 + x_4$ we have the narrowing sequence $x_1 + x_2 \rightsquigarrow_\tau x_3 + x_4$, using a variant of the first rule $\mathsf{0} + x \to x$. Finally, there is an mgu $\mu = \{x_1 \mapsto \mathsf{0}, x_2 \mapsto x_3 + x_4\}$ such that $u\sigma\mu = \mathsf{f}(\mathsf{0}, x_3 + x_4) = u\tau\mu$. Moreover, $x_1\sigma\mu = \mathsf{0}$ and $x_3\tau\mu = x_3$ are two different normal forms. Hence $\mathcal{R}_{262}$ is non-confluent by Lemma 7.

3 Implementation

Starting from its first participation in the confluence competition (CoCo)[3] in 2014 ConCon 1.2.0.3 came equipped with some non-confluence heuristics. Back then it only used Lemmas 1 and 3 and had no support for certification of the output. In the next two years (ConCon 1.3.0 and 1.3.2) we focused on other developments [5, 6, 9, 10] and nothing changed for the non-confluence part. For this year's CoCo (2017) we have added Lemma 7 employing conditional narrowing to ConCon 1.5 and the output of all of the non-confluence methods is now certifiable by CeTA.

Our implementation of Lemma 1 takes an unconditional rule $\rho\colon \ell \to r$, a substitution $\sigma = \{x \mapsto y\}$ with $x \in \mathcal{V}(r) \setminus \mathcal{V}(\ell)$ and y fresh w.r.t. ρ and builds the non-joinable fork $r \;{}_\rho\!\leftarrow \ell \to_\rho r\sigma$.

For Lemma 3 we have three concrete implementations that consider an overlap from which an unconditional CP $s \approx t$ arises: The first of which just takes this overlap and then checks that s and t are two different normal forms with respect to $\mathcal{R}_\mathsf{u}$. The second employs the tcap-function to check for non-joinability, that is, it checks whether $\mathsf{tcap}(s)$ and $\mathsf{tcap}(t)$ are not unifiable. The third makes a special call to the TRS confluence checker CSI [12] providing the underlying TRS $\mathcal{R}_\mathsf{u}$ as well as the unconditional CP $s \approx t$ where all variables in s and t have been replaced by fresh constants. We issue the following command:

```
csi -s '(nonconfluence -nonjoinability -steps 0 -tree)[30]' -C RT
```

The strategy '`(nonconfluence -nonjoinability -steps 0 -tree)[30]`' tells CSI to check non-joinability of two terms using tree automata techniques. Here '`-steps 0`' means that CSI does not rewrite the input terms further before checking non-joinability (this would be unsound in our setting). The timeout is set to 30 seconds. To encode the two terms for which we want

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

to check non-joinability in the input we set CSI to read relative-rewriting input ('`-C RT`'). We provide $\mathcal{R}_u$ in the usual Cops-format and add one line for the CP $s \approx t$ where its "grounded" left- and right-hand sides are related by '`->=`', that is, we encode it as a relative-rule. This is necessary to distinguish the unconditional CP from the rewrite rules.

Now, for an implementation of Lemma 7 we have to be careful to respect the freshness requirement of the variables in the used rule for every narrowing step with respect to all the previous terms and rules. The crucial point is to efficiently find the two narrowing sequences, to this end we first restrict the set of terms from which to start narrowing. As a heuristic we only consider the left-hand sides of rules of the CTRS under consideration. Next we also prune the search space for narrowing. Here we restrict the length of the narrowing sequences to at most three. In experiments on Cops allowing sequences of length four or more did not yield additional non-confluence proofs but slowed down the tool significantly to the point where we lost other proofs. Further, we also limit the recursion depth of conditional narrowing by restricting the level (see the definition of the conditional rewrite relation in the Preliminaries) to at most two. Again, we set this limit as tradeoff after thorough experiments on Cops. Finally, we use Property 6 to translate the forking narrowing sequences into forking conditional rewriting sequences. In this way we generate a lot of forking sequences so we only use fast methods, like non-unifiability of the tcap's of the endpoints or that they are different normal forms, to check for non-joinability of the endpoints. Calls to CSI are to expensive in this context.

4 Certification

Certification is quite similar for all of the described methods. We have to provide a non-confluence witness, that is, a non-joinable fork. So besides the CTRS $\mathcal{R}$ under investigation we also need to provide the starting term s, the two endpoints of the fork t and u, as well as, certificates for $s \to_{\mathcal{R}}^{+} t$ and $s \to_{\mathcal{R}}^{+} u$, and a certificate that t and u are not joinable. For the forking rewrite sequences we reuse a recent formalization of ours to build the certificates (see [7]). We also want to stress that because of Property 6 we did not have to formalize conditional narrowing because going from narrowing to rewrite sequences is already done in ConCon and in the certificate only the rewrite sequences show up. For the non-joinability certificate of t and u there are three options: either we state that t and u are two different normal forms or that $\mathsf{tcap}(t)$ and $\mathsf{tcap}(u)$ are not unifiable; both of these checks are performed within CeTA; or, when the witness was found by an external call to CSI, we just include the generated non-joinability certificate.

5 Experiments

We tested the above non-confluence methods on all 129 oriented CTRSs from Cops (as of 2017-06-27). Most of those are 3-CTRSs but there are also four 4-CTRSs. On the whole ConCon 1.5 and CeTA are able to certify non-confluence of 42 systems (including all 4-CTRSs). Last year's version of ConCon can show non-confluence of 30 of the same 129 CTRSs only using the implementation of Lemmas 1 and 3. By first removing infeasible rules, a feature which we have also recently implemented in ConCon, we gain another system (271, see Example 4). Another new feature is inlining of conditions (see [7]) which gives two more systems (351, 353). Finally, with the help of conditional narrowing (Lemma 7) we gain another 9 systems (272, 328, 330, 352, 391, 404, 410, 411, 524). In contrast ConCon 1.5 succeeds in showing confluence of 60 systems (where 7 use methods that are not certifiable yet, like using Waldmeister to show infeasibility of CCPs). So from ConCon's perspective only 27 of the 129 oriented CTRSs of Cops

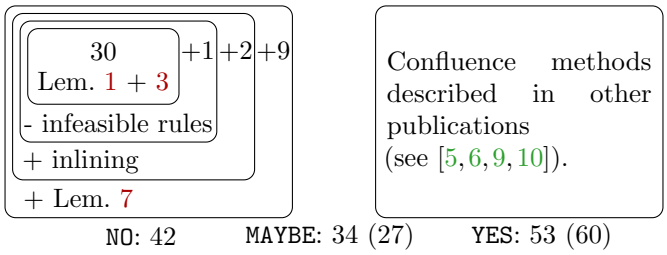


Figure 1: (Non)-confluence results on 129 oriented CTRSs.

are still open. These results are summarized in Figure 1.

Concerning future work we currently employ standard conditional narrowing in our implementation. Implementing basic conditional narrowing or LSE conditional narrowing should increase efficiency and maybe yield new NO-instances.

Acknowledgments. We thank Bertram Felgenhauer for providing the CSI-interface to check non-joinability of two terms. Also the first author wants to thank Vincent van Oostrom for valuable insights during the implementation of conditional narrowing.

References

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

[2] J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and Disproving Termination of Higher-Order Functions. In *Proc. 5th FROCOS*, volume 3717 of *LNCS*, pages 216–231. Springer, 2005. doi:10.1007/11559306_12.

[3] A. Middeldorp and E. Hamoen. Completeness Results for Basic Narrowing. *Appl. Algebra Eng. Commun. Comput.*, 5:213–253, 1994. doi:10.1007/BF01190830.

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

[5] C. Sternagel and T. Sternagel. Level-Confluence of 3-CTRSs in Isabelle/HOL. In *Proc. 4th IWC*, pages 28–32, 2015. arXiv:1602.07115.

[6] C. Sternagel and T. Sternagel. Certifying Confluence of Almost Orthogonal CTRSs via Exact Tree Automata Completion. In *Proc. 1st FSCD*, volume 51 of *LIPIcs*, pages 29:1–29:16. Dagstuhl, 2016. doi:10.4230/LIPIcs.FSCD.2016.29.

[7] C. Sternagel and T. Sternagel. Certifying Confluence of Quasi-Decreasing Strongly Deterministic Conditional Term Rewrite Systems. In *Proc. 26th CADE*, LNCS. Springer, 2017. To be published.

[8] T. Sternagel and A. Middeldorp. Conditional Confluence (System Description). In *Proc. Joint 25th RTA and 12th TLCA*, volume 8560 of *LNCS*, pages 456–465. Springer, 2014. doi:10.1007/978-3-319-08918-8_31.

[9] T. Sternagel and A. Middeldorp. Infeasible Conditional Critical Pairs. In *Proc. 4th IWC*, pages 13–17, 2015.

[10] T. Sternagel and C. Sternagel. Formalized Confluence of Quasi-Decreasing, Strongly Deterministic Conditional TRSs. In *Proc. 5th IWC*, pages 60–64, 2016. arXiv:1609.03341.

[11] R. Thiemann and C. Sternagel. Certification of Termination Proofs using CeTA. In *Proc. 22nd TPHOLs*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009. doi:10.1007/978-3-642-03359-9_31.

[12] H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – A Confluence Tool. In *Proc. 23rd CADE*, volume 6803 of *LNAI*, pages 499–505. Springer, 2011. doi:10.1007/978-3-642-22438-6_38.

A Semantic Criterion for Proving Infeasibility in Conditional Rewriting *

Salvador Lucas and Raúl Gutiérrez

DSIC, Universitat Politècnica de València, Spain

1 Introduction

In the literature about *confluence* and *termination* of (conditional) rewriting, a number of important issues can be investigated as *feasibility tests* specified as sequences $s_1 \to^* t_1, \ldots, s_n \to^* t_n$ of reachability conditions $s_i \to^* t_i$ where (in contrast to the usual reachability problems) the *instantiation* of variables in terms s_i and t_i is allowed for all $1 \leq i \leq n$. Given a *Conditional Term Rewriting System* (CTRS) $\mathcal{R}$, a sequence $s_1 \to^* t_1, \ldots, s_n \to^* t_n$ is $\mathcal{R}$-*feasible* if there is a substitution σ such that for all i, $1 \leq i \leq n$, $\sigma(s_i) \to^*_{\mathcal{R}} \sigma(t_i)$. In the realm of *oriented* CTRSs, with rules of the form $\ell \to r \Leftarrow c$ (with c usually written $s_1 \to t_1, \ldots, s_n \to t_n$ when the usual *rewriting semantics* for the evaluation of the conditions $s_i \to t_i$ is assumed), the *negation* of this property, i.e., the $\mathcal{R}$-*infeasibility* of a sequence $s_1 \to^* t_1, \ldots, s_n \to^* t_n$ as above, can be used to (i) disable the use of *conditional rules* $\ell \to r \Leftarrow c$ from $\mathcal{R}$ in reductions, (ii) discard *conditional critical pairs* $s \downarrow t \Leftarrow c$ in the analysis of *confluence* of CTRSs [11, 12, 13, 14], (iii) discard *conditional dependency pairs* $u \to v \Leftarrow c$ in the analysis of *operational termination* of CTRSs [9], or (iv) prove the *non-joinability* of terms s and t, possibly coming from conditional or unconditional critical pairs, in (C)TRSs: the $\mathcal{R}$-infeasibility of $s \to^* x, t \to^* x$ (with x not occurring in s or t) implies the non-joinability of s and t.

In this paper we apply the semantic approach to the analysis of properties of CTRSs in [5] to prove infeasibility: an $\mathcal{R}$-infeasibility problem is translated into the problem of finding a *model* $\mathcal{A}$ of the set of sentences $\overline{\mathcal{R}}$ representing the operational semantics of the CTRS $\mathcal{R}$ plus a sentence $\neg(\exists \vec{x})\ (s_1 \to^* t_1 \wedge \cdots \wedge s_n \to^* t_n)$ where $\to^*$ is viewed now as a predicate symbol in the underlying first-order language and can be freely interpreted in a first-order structure $\mathcal{A}$ [4]. If $\mathcal{A} \models \overline{\mathcal{R}} \cup \{\neg(\exists \vec{x})\ (s_1 \to^* t_1 \wedge \cdots \wedge s_n \to^* t_n)\}$ holds in the usual logical sense, i.e., the structure $\mathcal{A}$ satisfies all sentences in the right-hand side of '$\models$', then $s_1 \to^* t_1, \cdots, s_n \to^* t_n$ is $\mathcal{R}$-infeasible. In the examples discussed in this paper, such first-order structures have been synthesized using AGES, a tool for the automatic generation of models of first-order theories [3] which implements the methods described in [6]. Besides, we also introduce a number of auxiliary results which can be combined with this and other approaches in proofs of infeasibility, and hence in any application to prove confluence and operational termination of CTRSs.

2 Infeasibility problems

Borrowing [11, Definition 7.1.8(3)] for feasibility of Conditional Critical Pairs (CCPs, see [11, Definition 7.1.8(1)]), we introduce the following.

Definition 1. *Let $\mathcal{R}$ be a CTRS. A sequence $s_1 \to^* t_1, \ldots, s_n \to^* t_n$, where s_i and t_i are terms for all $1 \leq i \leq n$ is called a* feasibility sequence. *It is called* $\mathcal{R}$-feasible *if there is a substitution σ such that for all $1 \leq i \leq n$, $\sigma(s_i) \to^*_{\mathcal{R}} \sigma(t_i)$. Otherwise, it is called* $\mathcal{R}$-infeasible.

*Partially supported by the EU (FEDER), Spanish MINECO project TIN2015-69175-C4-1-R and GV project PROMETEOII/2015/013.

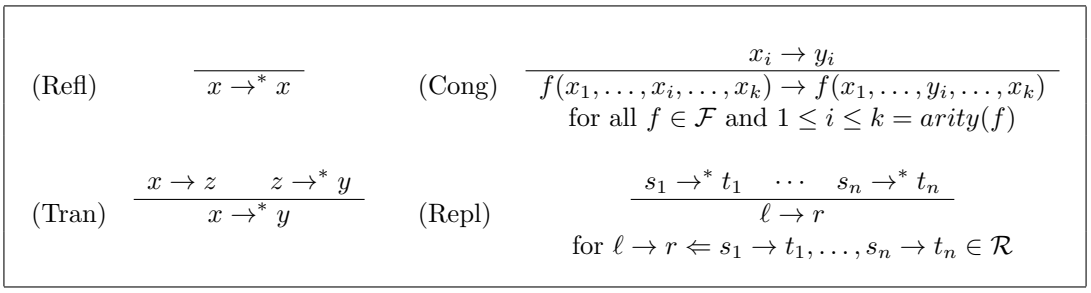


Figure 1: Inference rules for conditional rewriting with a CTRS $\mathcal{R}$ with signature $\mathcal{F}$

Following [7], in Definition 1 we write $s \to^*_{\mathcal{R}} t$ or $s \to_{\mathcal{R}} t$ for terms s and t iff there is a proof tree for $s \to^* t$ (resp. $s \to t$) using $\mathcal{R}$ in the inference system of Figure 1.

Remark 2. *All rules in the inference system in Figure 1 are* schematic *in that each inference rule* $\frac{B_1 \ \cdots \ B_n}{A}$ *can be used under any* instance $\frac{\sigma(B_1) \ \cdots \ \sigma(B_n)}{\sigma(A)}$ *of the rule by a substitution* σ*. For instance, (Repl) actually establishes that, for every rule* $\ell \to r \Leftarrow s_1 \to t_1, \ldots, s_n \to t_n$ *in the CTRS* $\mathcal{R}$*, every instance* $\sigma(\ell)$ *by a substitution* σ *rewrites into* $\sigma(r)$ *provided that, for each* $s_i \to t_i$*, with* $1 \le i \le n$*, the reachability condition* $\sigma(s_i) \to^* \sigma(t_i)$ *can be* proved*.*

Now, we can say (as usual) that a critical pair $s \downarrow t \Leftarrow c$ or a rule $\ell \to r \Leftarrow c$ where c is $s_1 \to t_1, \ldots, s_n \to t_n$ is $\mathcal{R}$-infeasible if $s_1 \to^* t_1, \ldots, s_n \to^* t_n$ is $\mathcal{R}$-infeasible. We can also see the usual joinability problem $s \downarrow t$, i.e, the existence of a term u such that $s \to^*_{\mathcal{R}} u$ and $t \to^*_{\mathcal{R}} u$, as an specific feasibility sequence.

Proposition 3. *Let* $\mathcal{R}$ *be a CTRS,* s, t *be terms, and* x *be a fresh variable not occurring in* s *or* t*. If* s *and* t *are joinable, then* $s \to^* x, t \to^* x$ *is* $\mathcal{R}$*-feasible. If* s *and* t *are* ground*, then the* $\mathcal{R}$*-feasibility of* $s \to^* x, t \to^* x$ *implies joinability of* s *and* t*.*

An immediate consequence of the previous observation is that the $\mathcal{R}$-infeasibility of $s \to^* x, t \to^* x$ implies the non-joinability of s and t. Aoto has investigated methods for proving non-joinability of *ground* terms in TRSs [1]. Actually, Aoto makes the following interesting observation: for Term Rewriting Systems $\mathcal{R}$, the so-called *usable rules for reachability* $\mathcal{U}(\mathcal{R}, s)$ associated to a term s (roughly speaking an *overapproximation* of the set of rules that can be applied to s and then to any other term introduced by the application of a rule, see [1, Definition 3]) are the only ones we need to consider in any rewriting sequence starting from s, i.e., $s \to^*_{\mathcal{R}} t$ iff $s \to^*_{\mathcal{U}(\mathcal{R},s)} t$ (see [1, Lemma 4]). Actually, this holds as well for the usable rules $\mathcal{U}(\mathcal{R}, t)$ for CTRSs $\mathcal{R}$ and terms s introduced in [10, Definition 11]. Thus, we have the following easy corollary of this fact, where, given terms $s_1, \ldots, s_n$, we let $\mathcal{U}(\mathcal{R}, s_1, \ldots, s_n) = \bigcup_{i=1}^n \mathcal{U}(\mathcal{R}, s_i)$.

Proposition 4. *Let* $\mathcal{R}$ *be a CTRS and* $s_1 \to^* t_1, \ldots, s_n \to^* t_n$ *be a feasibility sequence. If the sequence is* $\mathcal{U}(\mathcal{R}, s_1, \ldots, s_n)$*-feasible, then it is* $\mathcal{R}$*-feasible. If terms* s_i *are* ground *for all* $1 \le i \le n$ *and the sequence is* $\mathcal{R}$-feasible*, then it is* $\mathcal{U}(\mathcal{R}, s_1, \ldots, s_n)$-feasible*.*

Thus, we can prove $\mathcal{R}$-infeasibility of critical pairs and rules by proving the $\mathcal{U}(\mathcal{R}, s_1, \ldots, s_n)$-*infeasibility* of the corresponding feasibility sequence $s_1 \to^* t_1, \ldots, s_n \to^* t_n$, provided that terms s_i are ground for all $1 \le i \le n$.

$$(\forall x)\ \ x \to^* x \qquad (4)$$
$$(\forall x,y,z)\ \ x \to y \wedge y \to^* z \Rightarrow x \to^* z \qquad (5)$$
$$(\forall x,y)\ \ x \to y \Rightarrow \mathsf{f}(x) \to \mathsf{f}(y) \qquad (6)$$
$$(\forall x,y)\ \ x \to y \Rightarrow \mathsf{g}(x) \to \mathsf{g}(y) \qquad (7)$$
$$\mathsf{a} \to \mathsf{b} \qquad (8)$$
$$\mathsf{f}(\mathsf{a}) \to \mathsf{b} \qquad (9)$$
$$(\forall x)\ \ \mathsf{f}(x) \to^* x \Rightarrow \mathsf{g}(x) \to \mathsf{g}(\mathsf{a}) \qquad (10)$$

Figure 2: First-order theory for $\mathcal{R}$ in Example 5

3 A semantic criterion for infeasibility

In the logic of CTRSs, with binary *predicates* $\to$ and $\to^*$, the (first-order) theory $\overline{\mathcal{R}}$ for a CTRS $\mathcal{R} = (\mathcal{F}, R)$ is obtained from the inference rules in Figure 1 by *specializing* $(Cong)_{f,i}$ for each $f \in \mathcal{F}$ and i, $1 \le i \le ar(f)$ and $(Repl)_\rho$ for all $\rho : \ell \to r \Leftarrow c \in R$. Inference rules $\frac{B_1 \ \cdots \ B_n}{A}$ become universally quantified *implications* $B_1 \wedge \cdots \wedge B_n \Rightarrow A$ [8, Section 2].

Example 5. *Consider the following CTRS $\mathcal{R}$ [2, page 46]:*

$$\mathsf{a} \to \mathsf{b} \qquad (1)$$
$$\mathsf{f}(\mathsf{a}) \to \mathsf{b} \qquad (2)$$
$$\mathsf{g}(x) \to \mathsf{g}(\mathsf{a}) \Leftarrow \mathsf{f}(x) \to x \qquad (3)$$

Figure 2 shows its associated theory $\overline{\mathcal{R}}$.

By a structure $\mathcal{A}$ for a first-order logic language we mean an interpretation of the function and predicate symbols of the language ($f, g, \ldots$ and $P, Q, \ldots$, respectively) as mappings $f^{\mathcal{A}}, g^{\mathcal{A}}, \ldots$ and relations $P^{\mathcal{A}}, Q^{\mathcal{A}}, \ldots$ on a given set (carrier) also denoted $\mathcal{A}$. Then, the usual interpretation of first-order formulas with respect to the structure is considered. A model for a set $\mathcal{S}$ of first-order sentences (i.e., formulas whose variables are all *quantified*) is just a structure that makes them all true, written $\mathcal{A} \models \mathcal{S}$ see [4].

Proposition 6 (Semantic criterion for infeasibility)**.** *Let $\mathcal{R}$ be a CTRS, $s_1 \to^* t_1, \ldots, s_n \to^* t_n$ be a feasibility sequence, and $\mathcal{A}$ be a structure with nonempty domain. If $\mathcal{A} \models \overline{\mathcal{R}} \cup \{\neg(\exists \vec{x})\ (s_1 \to^* t_1 \wedge \cdots \wedge s_n \to^* t_n)\}$, then the sequence is $\mathcal{R}$-infeasible.*

Proof. By contradiction. Assume that the sequence is $\mathcal{R}$-feasible. Then, there is a substitution σ such that, for all $1 \le i \le n$, $\sigma(s_i) \to^*_{\mathcal{R}} \sigma(t_i)$. Since $\mathcal{A}$ is a model of $\overline{\mathcal{R}}$, by correctness we have that, for all $1 \le i \le n$, $\mathcal{A} \models (\forall \vec{y_i})\ \sigma(s_i) \to^* \sigma(t_i)$, where $\vec{y_i}$ are the variables in $\mathcal{V}ar(\sigma(s_i)) \cup \mathcal{V}ar(\sigma(t_i))$. Therefore,

$$\mathcal{A} \models (\forall \vec{y})\ (\sigma(s_1) \to^* \sigma(t_1) \wedge \cdots \wedge \sigma(s_n) \to^* \sigma(t_n)) \qquad (11)$$

with $\vec{y}$ the variables in $\bigcup_{i=1}^n \mathcal{V}ar(\sigma(s_i)) \cup \mathcal{V}ar(\sigma(t_i))$. Hence, for all valuations $\nu : \vec{y} \to \mathcal{A}$, the interpretation of the universally quantified formula in (11) in the structure $\mathcal{A}$, i.e., $[\sigma(s_1) \to^* \sigma(t_1) \wedge \cdots \wedge \sigma(s_n) \to^* \sigma(t_n)]^{\mathcal{A}}_{\nu}$, is *true*. Let $\vec{x}$ be the variables in $\bigcup_{i=1}^n \mathcal{V}ar(s_i) \cup \mathcal{V}ar(t_i)$. Since $\mathcal{A}$ has a nonempty domain, given an arbitrary valuation $\nu : \vec{y} \to \mathcal{A}$, there is a valuation $\nu' : \vec{x} \to \mathcal{A}$ given by $\nu'(x) = [\sigma(x)]^{\mathcal{A}}_{\nu}$ for all variable x in $\vec{x}$, such that $[s_1 \to^* t_1 \wedge \cdots \wedge s_n \to^* t_n]^{\mathcal{A}}_{\nu'}$ is true. This contradicts our assumption $\mathcal{A} \models \neg(\exists \vec{x})\ (s_1 \to^* t_1 \wedge \cdots \wedge s_n \to^* t_n)$. □

In the following, we assume that the signature $\mathcal{F}$ of any CTRS to be used with Proposition 6 contains at least a (dummy) *constant symbol* so that any associated structure $\mathcal{A}$ has a nonempty domain. Models to be used in Proposition 6 for infeasibility checking can be automatically

generated from the CTRS $\mathcal{R}$ and sentence at stake, i.e., $\neg(\exists\vec{x})\,(s_1 \to^* t_1 \wedge \cdots \wedge s_n \to^* t_n)$, by using (as we do in the following examples) a tool like AGES [3].[1]

Example 7 (Infeasible rule)**.** *The following structure $\mathcal{A}$ over $\mathbb{N} - \{0\}$:*

$$\mathsf{a}^{\mathcal{A}} = 1 \qquad \mathsf{b}^{\mathcal{A}} = 2 \qquad \mathsf{f}^{\mathcal{A}}(x) = x + 1 \qquad \mathsf{g}^{\mathcal{A}}(x) = 1$$
$$x \to^{\mathcal{A}} y \Leftrightarrow y \geq x \qquad x\,(\to^*)^{\mathcal{A}}\,y \Leftrightarrow y \geq x$$

is a model of $\overline{\mathcal{R}} \cup \{\neg(\exists x)\,\mathsf{f}(x) \to^* x\}$ for $\mathcal{R}$ in Example 5 (see $\overline{\mathcal{R}}$ in Figure 2). Thus, rule (3) is proved $\mathcal{R}$-infeasible. Using this observation it is not difficult to see that $\mathcal{R}$ is operationally terminating*, i.e., no term t has an infinite proof tree using the inference system in Figure 1 [7].*

Example 8 (Infeasible critical pair)**.** *The following CTRS $\mathcal{R}$ [14, Example 23]*

$$\mathsf{g}(x) \to \mathsf{f}(x,x) \tag{12}$$
$$\mathsf{g}(x) \to \mathsf{g}(x) \Leftarrow \mathsf{g}(x) \to \mathsf{f}(\mathsf{a},\mathsf{b}) \tag{13}$$

has a conditional critical pair $\mathsf{f}(x,x) \downarrow \mathsf{g}(x) \Leftarrow \mathsf{g}(x) \to \mathsf{f}(\mathsf{a},\mathsf{b})$. The following structure $\mathcal{A}$ over the finite domain $\{0,1\}$:

$$\mathsf{a}^{\mathcal{A}} = 1 \qquad \mathsf{b}^{\mathcal{A}} = \mathsf{c}^{\mathcal{A}} = 0 \qquad \mathsf{f}^{\mathcal{A}}(x,y) = \begin{cases} x - y + 1 & \text{if } x \geq y \\ y - x + 1 & \text{otherwise} \end{cases}$$
$$\mathsf{g}^{\mathcal{A}}(x) = 1 \qquad x \to^{\mathcal{A}} y \Leftrightarrow x = y \qquad x\,(\to^*)^{\mathcal{A}}\,y \Leftrightarrow x \geq y$$

is a model $\overline{\mathcal{R}} \cup \{\neg(\exists x)\,\mathsf{g}(x) \to^* \mathsf{f}(\mathsf{a},\mathsf{b})\}$. Thus, the critical pair is infeasible. In [14, Example 23] this is proved by using unification tests together with a transformation. It is discussed that the the alternative tree automata techniques investigated in the paper do not *work for this example.*

Example 9 (Infeasible rule)**.** *Consider the following CTRS $\mathcal{R}$ [14, Example 17]*

$$\mathsf{h}(x) \to \mathsf{a} \tag{14}$$
$$\mathsf{g}(x) \to x \tag{15}$$
$$\mathsf{g}(x) \to \mathsf{a} \Leftarrow \mathsf{h}(x) \to \mathsf{b} \tag{16}$$
$$\mathsf{c} \to \mathsf{c} \tag{17}$$

The following structure $\mathcal{A}$ over $\mathbb{N}$:

$$\mathsf{a}^{\mathcal{A}} = 0 \qquad \mathsf{b}^{\mathcal{A}} = \mathsf{c}^{\mathcal{A}} = 1 \qquad \mathsf{g}^{\mathcal{A}}(x) = x + 2 \qquad \mathsf{h}^{\mathcal{A}}(x) = 0$$
$$x \to^{\mathcal{A}} y \Leftrightarrow x \geq y \qquad x\,(\to^*)^{\mathcal{A}}\,y \Leftrightarrow x \geq y$$

is a model of $\overline{\mathcal{R}} \cup \{\neg(\exists x)\ \mathsf{h}(x) \to^* \mathsf{b}\}$. Therefore, rule (16) is proved $\mathcal{R}$-infeasible. In [14, Example 17] this is proved by using tree automata techniques. It is also shown that the alternative technique investigated in the paper (the use of unification tests) does not *work in this case.*

Example 10 (Non-joinable terms)**.** *Consider the following CTRS $\mathcal{R}$ [11, Example 7.3.3]:*

$$\mathsf{a} \to \mathsf{b} \tag{18}$$
$$\mathsf{f}(x) \to \mathsf{c} \Leftarrow x \to \mathsf{a} \tag{19}$$

Although there is no critical pair, the system is not (locally) confluent because $\mathsf{f}(\mathsf{a}) \to_{\mathcal{R}} \mathsf{f}(\mathsf{b})$ and $\mathsf{f}(\mathsf{a}) \to_{\mathcal{R}} \mathsf{c}$ but c and $\mathsf{f}(\mathsf{b})$ are not *joinable. The following structure $\mathcal{A}$ over $\mathbb{N} \cup \{-1\}$:*

$$\mathsf{a}^{\mathcal{A}} = 0 \qquad \mathsf{b}^{\mathcal{A}} = -1 \qquad \mathsf{c}^{\mathcal{A}} = 1 \qquad \mathsf{f}^{\mathcal{A}}(x) = x + 1$$
$$x \to^{\mathcal{A}} y \Leftrightarrow x = y \qquad x\,(\to^*)^{\mathcal{A}}\,y \Leftrightarrow x = y$$

is a model of $\overline{\mathcal{U}(\mathcal{R},\mathsf{f}(\mathsf{b}),\mathsf{c})} \cup \{\neg(\exists x)\,(\mathsf{f}(\mathsf{b}) \to^* x \wedge \mathsf{c} \to^* x)\}$, where $\mathcal{U}(\mathcal{R},\mathsf{f}(\mathsf{b}),\mathsf{c}) = \{(19)\}$. Therefore, $\mathsf{f}(\mathsf{b})$ and c are proved not joinable.

[1] In AGES the universally quantified version $(\forall\vec{x})\neg(s_1 \to^* t_1 \wedge \cdots \wedge s_n \to^* t_n)$ of $\neg(\exists\vec{x})(s_1 \to^* t_1 \wedge \cdots \wedge s_n \to^* t_n)$ is used; actually, only $\neg(s_1 \to^* t_1 \wedge \cdots \wedge s_n \to^* t_n)$ is introduced (universal quantification is assumed).

4 Conclusions

We have presented a semantic approach to prove infeasibility in conditional rewriting. Interestingly, with such a semantic approach we could handle many examples coming from papers developing different specific techniques to deal with these problems. In particular, we could deal with all examples solved in [13, 14] (some of them reported in our examples above; note that these papers explore several *alternative* methods and, as reported by the authors, some of them *fail* in specific examples which require a different approach). We also dealt with all Aoto's examples in [1] in combination with his *usable rules* refinement (see Proposition 4).

We do not have a dedicated, fully automated 'infeasibility' checker yet. Instead we just encode the problem we want to deal with (e.g., infeasibility of a critical pair, or rule; or non-joinability) as an specific infeasibility sequence and then use AGES to apply Proposition 6. The generation of a model is completely automatic, but in order to *succeed* on a given problem, we often require to *manually* change special settings like the generation of piecewise interpretations (like in Example 8), usually 'expensive' and not part of the *default* configuration. Thus, further investigation developing heuristics for an efficient use of the technique, possibly in combination with other existing techniques, would be necessary.

References

[1] T. Aoto. Disproving Confluence of Term Rewriting Systems by Interpretation and Ordering. In *Proc. of FroCoS'13*, LNCS 8152:311-326, 2013.

[2] J. Giesl and T. Arts. Verification of Erlang Processes by Dependency Pairs. *Applicable Algebra in Engineering, Communication and Computing* 12:39-72, 2001.

[3] R. Gutiérrez, S. Lucas, and P. Reinoso. A tool for the automatic generation of logical models of order-sorted first-order theories. In *Proc. of PROLE'16*, pages 215-230, 2016.

[4] W. Hodges. Elementary Predicate Logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic* Volume 1, pages 1-131. Reidel Publishing Company, 1983.

[5] S. Lucas. A Semantic Approach to the Analysis of Rewriting-Based Systems. In Proc. of *LOPSTR'17*, to appear, 2017.

[6] S. Lucas and R. Gutiérrez. Automatic Synthesis of Logical Models for Order-Sorted First-Order Theories. *Journal of Automated Reasoning*, DOI 10.1007/s10817-017-9419-3, 2017.

[7] S. Lucas, C. Marché, and J. Meseguer. Operational termination of conditional term rewriting systems. *Information Processing Letters* 95:446–453, 2005.

[8] S. Lucas and J. Meseguer. Models for Logics and Conditional Constraints in Automated Proofs of Termination. In Proc. of *AISC'14*, LNAI 8884:9-20, 2014.

[9] S. Lucas and J. Meseguer. Dependency pairs for proving termination properties of conditional term rewriting systems. *Journal of Logical and Algebraic Methods in Programming*, 86:236-268, 2017.

[10] S. Lucas and J. Meseguer. Normal forms and normal theories in conditional rewriting. *Journal of Logical and Algebraic Methods in Programming*, 85(1):67-97, 2016.

[11] E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer-Verlag, Apr. 2002.

[12] T. Sternagel and A. Middeldorp. Conditional Confluence (System Description). In *Proc. of RTA-TLCA'14*, LNCS 8560:456-465, 2014.

[13] T. Sternagel and A. Middeldorp. Infefasible Conditional Critical Pairs. In *Proc. of IWC'15*, pages 13–18, 2014.

[14] C. Sternagel and T. Sternagel. Certifying Confluence Of Almost Orthogonal CTRSs Via Exact Tree Automata Completion. In *Proc. of FSCD'16*, LIPIcs 52, Article No. 85; pp. 85:1–85:16, 2016.

Detecting Useless Critical Pairs

Cyrille Chenavier

Université Paris Diderot

1 Introduction

The critical pair/completion (CPC) technique was initiated in the mid sixties in three separated areas: theorem proving [12], polynomial ideal theory [3] and term rewriting [9]. For theoretical and practical reasons, improvements of CPC algorithms were developed in two main directions. The first one concerns strategies for selecting critical pairs. In [10], strategies consisting in adding the new critical pairs in a stack or in queue are investigated: the first one can fail even if another strategy succeeds whereas the second one always succeeds in the same situation. Other strategies consist in reducing critical pairs in parallel [1] and have shown to be efficient since the previously intractable cyclic 9 problem is solved using such a strategy in [6]. The second direction of improvement consists in finding criteria for detecting useless critical pairs. Buchberger introduced such a criterion in the context of polynomial ideal theory in [4] which was adapted to term rewriting systems in [13].

The presented work concerns the detection of useless critical pairs of rewriting systems whose underlying set of terms is a vector space. We are studying such rewriting systems since the theory of Gröbner bases concerns rewriting in a large class of algebraic structures (polynomial, tensor or Lie algebras, operads, invariant rings...) and we want that our framework generalises these various structures. For these structures, several criteria based on the so-called *syzygies* were introduced [7, 8, 11] for avoiding useless critical pairs during completion. As shown in [2], the computation of syzygies does not only enable us to reject critical pairs but to reject *useless reductions*. By useless reductions, we mean that all the critical pairs created from these reductions are useless. The downside of this approach is that useless reductions cannot be used to reduce terms into normal forms.

In this work, we introduce a lattice criterion for reducing the number of examinations of critical pairs. For that, we consider rewrite relations $\longrightarrow$ on a vector space V which admit decompositions

$$\longrightarrow = \bigcup_{i\,=\,1}^{n} \longrightarrow_i,$$

where each $\longrightarrow_i$ is also a rewrite relation on V. We propose an incremental completion procedure, that is we complete successively

$$\longrightarrow_{\leq i} = \bigcup_{j\,=\,1}^{i} \longrightarrow_j .$$

If $\longrightarrow_{\leq i}$ is already completed, we are looking for useless reductions of the form $v_1 \longrightarrow_{i+1} v_2$. In order to detect such reductions, we introduce in 2.2 the notion of *reduction operator*, which provide functional descriptions of rewriting systems. We recall in 2.3 that reduction operators admit a lattice structure, whose upper bound is written $\vee$. Letting N_i be the reduction operator normalising every element for the completion of $\longrightarrow_{\leq i}$ and T_{i+1} the reduction operator corresponding to the rewrite relation $\longrightarrow_{i+1}$, our criterion rejects the reductions $v_1 \longrightarrow_{i+1} v_2$ such that v_1 does belong to the image of $N_i \vee T_{i+1}$. In Section 3, we illustrate our criterion with a complete example.

B. Accatoli and B. Felgenhauer (eds.); IWC 2017, pp. 21–26

2 Reduction Operators

2.1. Notations. We fix a well-ordered set $(G, \ )$ and a commutative field $\mathbb{K}$. Every vector v of the vector space $\mathbb{K}G$ spanned by G admits a greatest element, written $\lg(v)$, in its decomposition with respect to G. We extend the order $<$ on G into an order on $\mathbb{K}G$ defined by $v_1 < v_2$ if $v_1 = 0$ and $v_2 \neq 0$ or if $\lg(v_1) < \lg(v_2)$.

2.2. Definition. A linear endomorphism T of $\mathbb{K}G$ is called a *reduction operator relative to* $(G, <)$ if it is a projector and if for every $g \in G$, we have $T(g) \leq g$. We write $\mathbf{RO}(G, <)$ the set of reduction operators relative to $(G, <)$ and for every $T \in \mathbf{RO}(G, <)$, we write

$$\mathrm{NF}(T) = \Big\{ g \in G \mid T(g) = g \Big\}.$$

2.3. Lattice Structure. Recall from [5, Proposition 2.1.14] that the map

$$\begin{aligned} \ker : \mathbf{RO}(G, <) &\longrightarrow \Big\{ \text{subspaces of } \mathbb{K}G \Big\}, \\ T &\longmapsto \ker(T) \end{aligned}$$

is a bijection. Given a subspace V of $\mathbb{K}G$, we write $\ker^{-1}(V)$ the unique reduction operator with kernel V. Then, $(\mathbf{RO}(G, <), \preceq, \wedge, \vee)$ is a lattice where

i. $T_1 \preceq T_2$ if $\ker(T_2) \subseteq \ker(T_1)$,

ii. $T_1 \wedge T_2 = \ker^{-1}(\ker(T_1) + \ker(T_2))$,

iii. $T_1 \vee T_2 = \ker^{-1}(\ker(T_1) \cap \ker(T_2))$.

2.4. Confluence. Let $F \subset \mathbf{RO}(G, <)$. We write

$$\mathrm{NF}(F) = \bigcap_{T \in F} \mathrm{NF}(T) \text{ and } \wedge F = \ker^{-1}\left(\sum_{T \in F} \ker(T) \right).$$

The set F is said to be *confluent* if we have the equality $\mathrm{NF}(\wedge F) = \mathrm{NF}(F)$. Recall from [5, Corollary 2.3.9] that F is confluent if and only if the reduction relation defined by

$$v \underset{F}{\longrightarrow} T(v),$$

for every $T \in F$ and every $v \notin \mathrm{im}(T)$, is confluent.

2.5. Completion. Let F be a subset of $\mathbf{RO}(G, <)$. A *completion* of F is a set $F' \subset \mathbf{RO}(G, <)$ such that

i. F' is confluent,

ii. $F \subseteq F'$ and $\wedge F' = \wedge F$.

We let

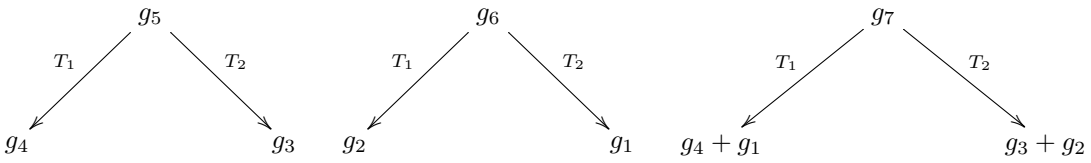
$$C^F = (\wedge F) \vee \left(\vee \overline{F} \right),$$

where $\vee \overline{F}$ is equal to $\ker^{-1}(\mathbb{K}\mathrm{NF}(F))$. Recall from [5, Theorem 3.2.6] that $F \cup \left\{ C^F \right\}$ is a completion of F.

2.6. Example. We consider $G = (g_1 < g_2 < g_3 < g_4 < g_5 < g_6 < g_7)$. We let $P = (T_1,\ T_2)$, where

$$T_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } T_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

where the matrices are considered with respect to the basis G. The pair P represents the following reductions:



We have

$$C^P = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

2.7. Remark. In the previous example, we remark that C^P is equal to $C^{P'}$, where P' is the pair obtained by considering the restrictions of T_1 and T_2 to the vector space spanned by $G \setminus \{g_7\}$. Hence, this example shows that there exist elements of G that we can avoid during a completion procedure. Our purpose in the sequel is to provide a criterion using the lattice structure to detect these useless elements.

2.8. Restrictions and Extensions of Reduction Operators. Let $P = (T_1,\ T_2)$ be a pair of reduction operators relative to $(G,\ <)$. We consider the pair $P' = (T_1',\ T_2')$ of reduction operators relative to $\left(\mathrm{NF}\left(T_1 \vee T_2\right),\ <\right)$ defined by $T_i'(g) = T_i(g)$ for every $g \in \mathrm{NF}\left(T_1 \vee T_2\right)$ and $i = 1$ or 2.

Let $\tilde{G}$ be a subset of G and let $T \in \mathbf{RO}\left(\tilde{G},\ <\right)$. Let $\overline{T} \in \mathbf{RO}\left(G,\ <\right)$ defined by

$$\overline{T}(g) = \begin{cases} T(g), & \text{if } g \in \tilde{G} \\ g, & \text{otherwise} \end{cases}$$

for every $g \in G$.

2.9. Proposition. *Let $F = (T_1, \cdots, T_n)$ be a finite set of reduction operators. For every $2 \leq i \leq n$, we let $P_i = (T_1 \wedge \cdots \wedge T_{i-1}, T_i)$. Then,*

$$F \cup \left\{ \overline{C^{P_2'}} \wedge \cdots \wedge \overline{C^{P_n'}} \right\},$$

is a completion of F.

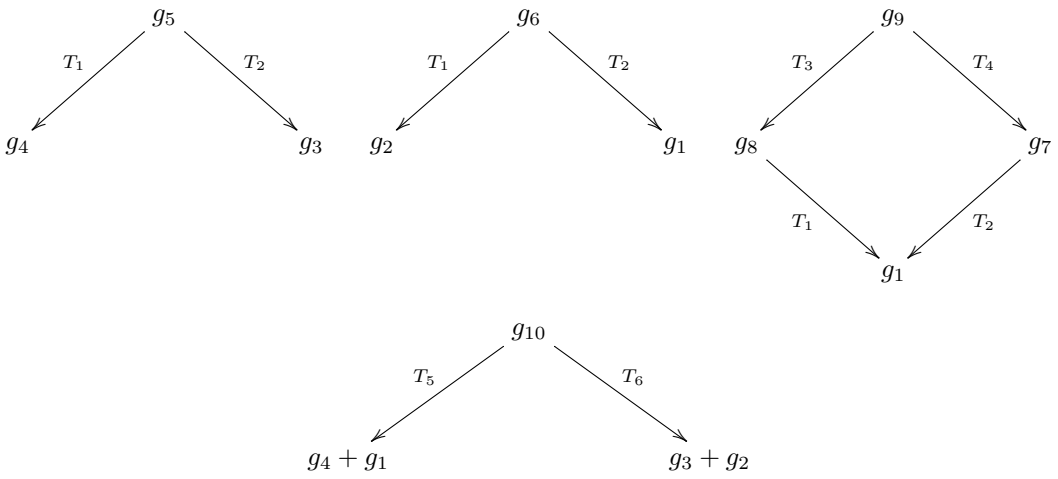
2.10. Remark.

i. The previous proposition means that the reductions $g \xrightarrow[T_i]{} g$ are useless for completion whenever g is a normal form for $(T_1 \wedge \cdots \wedge T_{i-1}) \vee T_i$.

ii. In the previous proposition, we could replace the construction $\overline{C^{P_2'}} \wedge \cdots \wedge \overline{C^{P_n'}}$ by $\overline{C^{F'}}$, where F' is obtained by considering the restrictions of elements of F to the union of the sets $\mathrm{NF}\left((T_1 \wedge \cdots \wedge T_{i-1}) \vee T_i\right)$. The construction $\overline{C^{P_2'}} \wedge \cdots \wedge \overline{C^{P_n'}}$ means that we complete successively completions of (T_1, T_2), (T_1, T_2, T_3), ..., $(T_1, \cdots, T_n) = F$. We illustrate this step by step construction in the next section.

3 Example

3.1. Initial Data. Consider $G = (g_1 < g_2 < g_3 < g_4 < g_5 < g_6 < g_7 < g_8 < g_9 < g_{10})$ and $F = (T_1, T_2, T_3, T_4, T_5, T_6)$ represented by the following reductions:



3.2. Organisation. We compute C^F step by step. We initialize the completion with

$$C = \mathrm{Id}_{\mathbb{K}G}.$$

At each step i, we select the elements g of G reducible both for $T_1 \wedge \cdots \wedge T_{i-1}$ and by T_i. If g is reducible by $(T_1 \wedge \cdots \wedge T_{i-1}) \vee T_i$, we do not consider g in the completion process.

We do not give the details of the computations. They were treated using the online implementation of reduction operators available on the website `www.irif.fr/~chenavier`.

3.3. Step 1. We consider $P_2 = (T_1, T_2)$. We have two elements of G reducible by T_1 and T_2: g_5 and g_6. Moreover, $T_1 \vee T_2$ is equal to the identity matrix of $\mathbb{K}G$, so that we need to consider both g_5 and g_6. We obtain that $C = C^{P_2}$ maps g_4 to g_3 and g_2 to g_1.

3.4. Step 2. We consider the pair $P_3 = (T_1 \wedge T_2, T_3)$. There is no element reducible by $T_1 \wedge T_2$ and by T_3, so that there is no completion at this step.

3.5. Step 3. We consider $P_4 = (T_1 \wedge T_2 \wedge T_3, T_4)$. There is one element reducible both by $T_1 \wedge T_2 \wedge T_3$ and T_4: g_9. Moreover, $(T_1 \wedge T_2 \wedge T_3) \vee T_4$ maps g_9 to g_7, i.e., $\mathrm{Red}\,((T_1 \wedge T_2 \wedge T_3) \vee T_4)$ is reduced to $\{g_9\}$. Hence, there is no completion at this step.

3.6. Step 4. We consider $P_5 = (T_1 \wedge T_2 \wedge T_3 \wedge T_4, T_5)$. There is no element reducible by $T_1 \wedge T_2 \wedge T_3 \wedge T_4$ and by T_5, so that there is no completion at this step.

3.7. Step 5. We consider $P_6 = (T_1 \wedge T_2 \wedge T_3 \wedge T_4 \wedge T_5, T_6)$. There is one element reducible both by $T_1 \wedge T_2 \wedge T_3 \wedge T_4 \wedge T_5$ and T_6: g_{10}. Moreover, $(T_1 \wedge T_2 \wedge T_3 \wedge T_4 \wedge T_5) \vee T_6$ maps g_{10} to $g_3 + g_2$, that is $\mathrm{Red}\,((T_1 \wedge T_2 \wedge T_3 \wedge T_4 \wedge T_5) \vee T_6)$ is reduced to $\{g_{10}\}$. Hence, there is no completion at this step and the completion terminates with

$$C^F = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

References

[1] Beatrice Amrhein, Reinhard Bündgen, and Wolfgang Küchlin. Parallel completion techniques. In *Symbolic rewriting techniques (Ascona, 1995)*, volume 15 of *Progr. Comput. Sci. Appl. Logic*, pages 1–34. Birkhäuser, Basel, 1998.

[2] Alberto Arri and John Perry. The F5 criterion revised. *J. Symbolic Comput.*, 46(9):1017–1029, 2011.

[3] Bruno Buchberger. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal. *Universitat Innsbruck, Austria, Ph. D. Thesis*, 1965.

[4] Bruno Buchberger. A criterion for detecting unnecessary reductions in the construction of Gröbner-bases. In *Symbolic and algebraic computation (EUROSAM '79, Internat. Sympos., Marseille, 1979)*, volume 72 of *Lecture Notes in Comput. Sci.*, pages 3–21. Springer, Berlin-New York, 1979.

[5] Cyrille Chenavier. Reduction Operators and Completion of Rewriting Systems. *J. Symbolic Comput.*, 2017.

[6] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *J. Pure Appl. Algebra*, 139(1-3):61–88, 1999. Effective methods in algebraic geometry (Saint-Malo, 1998).

[7] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 75–83 (electronic). ACM, New York, 2002.

[8] Rüdiger Gebauer and H. Michael Möller. On an installation of Buchberger's algorithm. *J. Symbolic Comput.*, 6(2-3):275–286, 1988. Computational aspects of commutative algebra.

[9] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967)*, pages 263–297. Pergamon, Oxford, 1970.

[10] Wolfgang Küchlin. A theorem-proving approach to the knuth-bendix completion algorithm. *Computer Algebra*, pages 101–108, 1982.

[11] Hans-Michael Möller, Teo Mora, and Carlo Traverso. Gröbner bases computation using syzygies. In *Papers from the international symposium on Symbolic and algebraic computation*, pages 320–328. ACM, 1992.

[12] John A. Robinson. A machine-oriented logic based on the resolution principle. *J. Assoc. Comput. Mach.*, 12:23–41, 1965.

[13] Franz Winkler. Reducing the complexity of the Knuth-Bendix completion algorithm: a "unification" of different approaches. In *EUROCAL '85, Vol. 2 (Linz, 1985)*, volume 204 of *Lecture Notes in Comput. Sci.*, pages 378–389. Springer, Berlin, 1985.

Coherence in Non-Terminating Linear Polygraphs

Clément Alleaume

Université de Lyon, Institut Camille Jordan, CNRS UMR 5208
Université Claude Bernard Lyon 1, 43, boulevard du 11 novembre 1918,
69622 Villeurbanne cedex, France
clement.alleaume@univ-st-etienne.fr

1 Introduction

In equational theories, coherence rules guarantee computational properties of the axioms of the theory. In computer science, logic and algebra, solving a coherence problem amounts to finding the coherence rules of a given theory. Several works were developed to study coherence problems by rewriting methods [7]. These methods use Squier's Theorem [12], which allows the construction of a coherent presentation of a monoid from a convergent rewriting system presenting this monoid. More precisely, a family of confluence diagrams associated to all critical pairs for a convergent presentation of a monoid generates the homotopical syzygies for this monoid. These syzygies express coherence relations between parallel reduction paths. The termination condition for the construction of coherent presentations has been weakened in [3, 4]. Squier's Theorem finds many applications in representation theory, like the study of Artin monoids [5] or plactic monoids [8].

The tools of linear rewriting can also be used to solve coherence problems for algebras or linear categories. Gröbner bases are an important example of such a tool, see [11]. Termination for Gröbner bases comes from monomial orders. These orders yield terminating sets of rules for algebras which can be completed into confluent ones by using Buchberger's procedure, making convergent presentations. Those convergent presentations can then be used to construct coherent presentations of algebras by choosing a family of confluence diagrams of the critical pairs. In [6], the condition of having a monomial order is weakened into termination without compatibility with a monomial order. This result corresponds to a linear version of Squier's Theorem. However convergence is not always easy to obtain, like in the case of Hecke algebras [9] or others important families of algebras. Thus, we want to weaken the convergence condition to keep only confluence and quasi-termination, a condition weaker than termination but sufficient to treat the case of Hecke algebras.

For example linear Squier's Theorem can be used to construct a coherent presentation for the non commutative algebra presented by the linear polygraph with one 0-cell, three 1-cells x, y and z, and one 2-cell $xyz \Rightarrow x^3 + y^3 + z^3$. This presentation yields a terminating and confluent rewriting system. Thus, by linear Squier's Theorem, we can construct a coherent presentation for this algebra. Now, consider the algebra $\mathbf{A}$ presented by the linear polygraph Σ with one 0-cell, two 1-cells x and y and one 2-cell $xy \Rightarrow x^2 + y^2$. The linear polygraph Σ is not terminating because of the rewriting sequence $x^2y \Rightarrow x^3 + xy^2 \Rightarrow x^3 + x^2y + y^3$. This prevents us from finding a coherent presentation of $\mathbf{A}$ with the given linear polygraph by using linear Squier's Theorem.

Some algebras, like Hecke algebras, do not admit any finite convergent presentation on a fixed set of generators, see [10]. To construct coherent presentations of those algebras, one needs to weaken the termination hypothesis of the linear Squier's Theorem into quasi-termination and choose orientations of rules which make confluence trivial at the cost of losing termination. This

B. Accatoli and B. Felgenhauer (eds.); IWC 2017, pp. 27–32

is in essence our main result 4.5. However, the presence of a linear structure makes the problem harder than in the set-wise setting.

In section 2, we recall the notions of linear polygraphs presented in [2]. We introduce multiple notions of decreasingness for linear polygraphs. We then present in section 3 a criterion of local confluence from quasi-termination and confluence of the critical branchings 3.1. In section 4, we recall from [1] the notion of coherent presentation for linear categories and introduce the different notions of loops for linear polygraphs. We then prove our main result 4.5. We illustrate this result by two examples. The first example is a case of Hecke algebra in which we orient a rule to obtain quasi-termination. The second example treats the case of $\mathbf{A} = \langle x, y \mid xy = x^2 + y^2 \rangle$.

2 Linear labelled two-dimensional polygraphs

2.1. Linear $(2,2)$-polygraphs and rewriting. Let R be a commutative ring. An R-*linear* $(1,1)$-*category*, or linear $(1,1)$-category if not ambiguous, is a 1-category enriched in R-modules. A *(*R*-)linear* $(2,2)$-*polygraph* is a triple $\Sigma = (\Sigma_0, \Sigma_1, \Sigma_2)$, where (Σ_0, Σ_1) is a 1-polygraph and Σ_2 is a globular extension of the free R-linear category Σ_1^ℓ over Σ_1. The (R-)linear $(2,2)$-polygraph Σ is called *left-monomial* if all elements of Σ_2 have a source in Σ_1^*, for such a polygraph *monomial* is a 1-cell of Σ_1^*. We will respectively use the terms *linear category* and *linear polygraph* in place of linear $(1,1)$-category and linear $(2,2)$-polygraph. From now on, we assume that all linear polygraphs are left-monomial.

Let Σ be a linear polygraph. A *rewriting step* of Σ is a 2-cell of the form $w + \lambda u \varphi v$ where u and v are monomials, φ is a 2-cell of Σ_2, w is a linear combination of monomials which does not contain $u s_1(\varphi) v$ and λ is a non zero scalar. We say that Σ is *quasi-terminating* if any infinite rewriting path of Σ contains infinitely many occurrences of a same 1-cell.

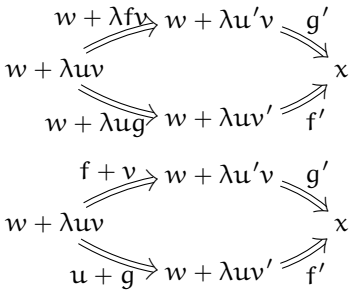
A linear polygraph Σ is *(scalar) exponentiation free* if no monomial m of Σ^ℓ can be rewritten into $\lambda m + f$ for some scalar λ other than 0 or 1 and some non zero 1-cell f which does not contain m in its monomial decomposition. Note that if Σ is quasi-terminating, exponentiation freedom is equivalent to the the fact that for every monomial m rewriting into a 1-cell f containing m in its monomial decomposition, we have $f = m$.

2.2. Labelled linear polygraphs. A *labelled linear polygraph* is the data of a linear polygraph Σ, a set W and a map ψ from Σ_{stp} to W. We say that ψ is *whisker-compatible* if for any rewriting steps f and g such that $\psi(f) \prec \psi(g)$, we have $\psi(u_1 f u_2 + v) \prec \psi(u_1 g u_2 + v)$ for any composable 1-cells u_1 and u_2 in Σ_1^ℓ and any 1-cell v of Σ_1^ℓ such that $u_1 f u_2 + v$ and $u_1 g u_2 + v$ are rewriting steps.

A *well-founded labelled linear polygraph* is a data $(\Sigma, W, \prec, \psi)$ made of a linear 2-polygraph Σ, a set W, a well-founded order $\prec$ on W and a map $\psi : \Sigma_{stp} \longrightarrow W$. The map ψ is called a *well-founded labelling* of Σ and associates to a rewriting step f a *label* $\psi(f)$. Given a rewriting sequence f, we denote by $L^W(f)$ the set of labels of rewriting steps in f. Note that two distinct rewriting sequences f and g can correspond to a same 2-cell in Σ_2^ℓ despite $L^W(f)$ and $L^W(g)$ being distinct. We say a linear polygraph is *strictly decreasing* if there exists a well-founded labelling $(W, \prec, \psi)$ of Σ such that all local branchings (f, g) of Σ can be completed into a confluence diagram $(f \cdot f', g \cdot g')$ such that all elements of $L^W(f')$ are lower than $\psi(f)$ and all elements of $L^W(g')$ are lower than $\psi(g)$. Note that strict decreasingness is a particular case of the decreasingness property defined in [13].

2.3. Local branchings of a linear polygraph. An aspherical branching of Σ is a local branching (f, g) such that $f = g$. A *Peiffer branching* of Σ is a local branching $(w+\lambda fv, w+\lambda ug)$ with 1-source uv where u, v are monomials,f, g are in Σ_{stp}, λ is a non zero scalar and w is a linear combination of monomials which does not contain uv. An *additive branching* of Σ is a local branching $(f + v, u + g)$ with 1-source $u + v$ where u and v do not have any common monomial in their decomposition and f, g are in Σ_{stp}. An *overlapping branching* of Σ is a local branching that is not aspherical, Peiffer or additive. An overlapping branching is called a *critical branching* if its source is a minimal monomial.

Let Σ be a linear polygraph and let Γ be a globular extension of the linear $(2, 1)$-category Σ^ℓ. The linear polygraph Σ is *(strictly) Peiffer decreasing with respect to* Γ if there exists a well-founded labelling $(W, \prec, \psi)$ such that Σ is decreasing with respect to the labelling $(W, \prec, \psi)$, for any Peiffer branching $(w + \lambda fv, w + \lambda ug)$, there exists a (strictly) decreasing confluence diagram such that$(w + \lambda fv) \star_1 f' \equiv_\Gamma (w + \lambda ug) \star_1 g'$ and for any additive branching $(f+v, u+g)$, there exists a (strictly) decreasing confluence diagram such that $(f + v) \star_1 f' \equiv_\Gamma (u + g) \star_1 g'$.



3 Decreasingness and confluence

A decreasing linear polygraph is confluent. This result has the same proof than the abstract rewriting result of [13]. Newman's lemma is also the same as the one for 2-polygraphs. However, the critical pair lemma is different than the one for 2-polygraphs as proved in [6].

3.1. Lemma. *Let Σ be a quasi-terminating exponentiation free linear polygraph. The two following properties hold:*

- *all additive branchings of Σ are confluent,*
- *for any 1-cell u of Σ_1^ℓ, if all critical branchings of Σ are confluent and Σ is locally confluent at every 1-cell v such that $v \prec u$, then, Σ is locally confluent at u.*

This Lemma can be proved using the same proof sketch that [6] by checking all families of local branchings. Aspherical branchings being always confluent, we prove that all additive and Peiffer branchings of Σ are confluent. We then prove that all overlapping branchings are confluent by using the confluence of critical branchings.

In this way, we prove the following Theorem by well-founded induction on the order $\prec$.

3.2. Theorem. *Let Σ be a quasi-terminating exponentiation free linear polygraph. Then Σ is locally confluent if and only if all its critical branchings are confluent.*

3.3. A counterexample without exponentiation freedom. Exponentiation freedom is a necessary condition to Lemma 3.1. Let us consider for example the algebra presented by the linear polygraph with generating 1-cells x, y, z, t and rules $xz \Rightarrow xy$, $yt \Rightarrow \lambda zt$. This linear polygraph is not exponentiation free if $\lambda \notin \{0, 1\}$. For example, if $\lambda = -1$, we have the infinite rewriting sequence $xyt \Rightarrow -xzt \Rightarrow -xyt \Rightarrow xzt \Rightarrow \dots$ or, if $\lambda = 2$ in a field of characteristic 0, we do not even have quasi-termination because of the infinite rewriting

sequence $xyt \Rightarrow 2xzt \Rightarrow 2xyt \Rightarrow 4xzt \Rightarrow \dots$. In this linear polygraph, the additive branching of source $xyt + xzt$ is not confluent, see [6].

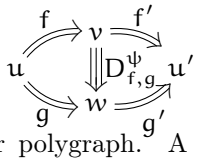
4 Coherence by decreasingness in linear categories

4.1. Coherent presentation of a linear category. An R-*linear* $(2,2)$-*category*, or linear $(2,2)$-category if not ambiguous, is a 1-category enriched in R-linear categories. A *(R-)linear polygraph* is a triple $\Sigma = (\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3)$, where $(\Sigma_0, \Sigma_1, \Sigma_2)$ is a linear polygraph and Σ_3 is a globular extension of the free R-linear $(2,2)$-category Σ_2^ℓ over Σ_2. A *coherent presentation* of a linear category $\mathcal{C}$ is a linear $(3,2)$-polygraph Σ such that the linear polygraph $(\Sigma_0, \Sigma_1, \Sigma_2)$ presents $\mathcal{C}$ and the quotient of the free $(2,2)$-linear category over $(\Sigma_0, \Sigma_1, \Sigma_2)$ by Σ_3 is an aspherical $(2,1)$-module.

4.2. Loops. A 2-*loop* in the linear $(2,2)$-category Σ_2^ℓ is a 2-cell f of Σ_2^ℓ such that $s_1(f) = t_1(f)$. Two 2-loops f and g in Σ_2^ℓ are *equivalent* if there exist a decomposition $f = f_1 \star_1 \dots \star_1 f_p$, where f_i is a rewriting step of Σ for any $1 \leqslant i \leqslant p$, and a circular permutation σ such that $g = f_{\sigma(1)} \star_1 \dots \star_1 f_{\sigma(p)}$. This defines an equivalence relation on 2-loops of Σ_2^ℓ. We denote by $\mathcal{L}(f)$ the equivalence class of a 2-loop f in Σ_2^ℓ for this relation. A 2-loop f in Σ_2^ℓ is *minimal with respect to* 1*-composition*, if any decomposition $f = g \star_1 h \star_1 k$ in Σ_2^ℓ with h a 2-loop implies that h is either an identity or equal to f. A 2-loop f in Σ_2^ℓ is *minimal by context*, if there is no decomposition $f = ugv + h$, where u and v are nonidentity 1-cells in Σ_1^*, g is a loop in Σ_2^ℓ and h is a nonzero loop. A 2-loop f in Σ_2^ℓ is *elementary* if it is minimal both with respect to 1-composition and by context.

4.3. Generating decreasing confluences. We denote by $\mathcal{E}(\Sigma)$ the set of equivalence classes of elementary 2-loops of Σ_2^ℓ. A *loop extension* of Σ is a globular extension of the $(2,1)$-category $\Sigma_2^\top$ made of a family of 3-cells $A_\alpha : \alpha \Rrightarrow 1_{s_1(\alpha)}$ indexed by exactly one α for each equivalence class in $\mathcal{E}(\Sigma)$.

Let (Σ, ψ) be a decreasing linear polygraph. A *family of generating decreasing confluences of* Σ *with respect to* ψ is a globular extension of the Σ_2^ℓ that contains, for every critical branching $(f, g) : u \Rightarrow (v, w)$ of Σ, exactly one 3-cell $D^\psi_{f,g}$ of the following form and where the confluence diagram $(f \cdot f', g \cdot g')$ is decreasing with respect to ψ.



4.4. Squier's decreasing completion. Let (Σ, ψ) be a decreasing linear polygraph. A *Squier's decreasing completion of* Σ *with respect to* ψ is a linear $(3,2)$-polygraph that extends the 2-polygraph Σ by a globular extension $\mathcal{O}(\Sigma, \psi) \cup \mathcal{L}(\Sigma)$ where $\mathcal{O}(\Sigma, \psi)$ is a chosen family of generating decreasing confluences with respect to ψ and $\mathcal{L}(\Sigma)$ is a loop extension of Σ defined in [4, 2.2.5.]. If (Σ, ψ) is a strictly decreasing 2-polygraph, a *strictly decreasing Squier's completion* is a Squier's decreasing completion, whose the generating decreasing confluences are required strict.

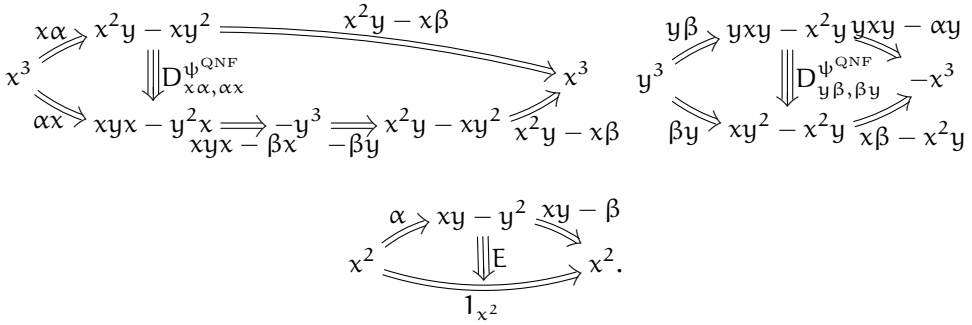
4.5. Theorem. *Let Σ be a quasi-terminating exponentiation free linear polygraph. Let $\mathcal{D}(\Sigma, \psi)$ be a strictly decreasing Squier's completion of Σ with respect to a labelling to the quasi-normal form (or QNF labelling) $(W, \psi, \prec)$. If ψ is whisker compatible, and ψ is strictly Peiffer decreasing with respect to $\mathcal{D}(\Sigma, \psi)$, then the strictly decreasing Squier's completion $\mathcal{D}(\Sigma, \psi)$ is a coherent presentation of the linear category presented by Σ.*

4.6. Examples. 1) The Hecke algebra of the monoid $\mathbf{B}_3^+$ is the $\mathbb{Z}[q, q^{-1}]$-algebra generated by two elements s and t subject to the relations $sts = tst$, $s^2 = (1-q)s$, $t^2 = (1-q)t$. We consider the presentation of this algebra by the following $\mathbb{Z}[q, q^{-1}]$-linear polygraph:

$$\Lambda(\mathbf{B}_3^+) = \big\langle s, t \;\big|\; sts \Rightarrow tst,\ tst \Rightarrow sts,\ s^2 \Rightarrow (1-q)s,\ t^2 \Rightarrow (1-q)t \big\rangle.$$

The linear polygraph $\Lambda(\mathbf{B}_3^+)$ is quasi-terminating. Let ψ^{QNF} be the QNF labelling of $\Lambda(\mathbf{B}_3^+)$ defined by choosing the quasi-normal forms as the linear combination of monomials of the form $q^n(sts)^N\nu$ where $(sts)^N\nu$ is a word not containing the letter q and defined as in [4, Example 2.4.10.]. This labelling is whisker compatible and strictly Peiffer decreasing with respect to the decreasing Squier's completion $\mathcal{D}(\Sigma, \psi^{\mathrm{QNF}})$ associated to ψ^{QNF}. This globular extension contains one 3-cell for each critical branching with the sources $ststst$, $tststs$, $stssts$, $tstst$, $ssts$, $stss$, $ttst$, $tstt$, sss, ttt, and one 3-cell associated to the only elementary 2-loop up to equivalence $sts \Rightarrow tst \Rightarrow sts$.

2) Let $\mathbb{K}$ be a field and $\mathbf{A}$ be the $\mathbb{K}$-algebra generated by two elements x and y subject to the relation $xy = x^2 + y^2$. The $\mathbb{K}$-linear polygraph $\Lambda = \big\langle x, y \;\big|\; \alpha : x^2 \Rightarrow xy - y^2,\ \beta : y^2 \Rightarrow xy - x^2 \big\rangle$. is a confluent and quasi-terminating presentation of $\mathbf{A}$. The linear polygraph Λ has only one elementary 2-loop up to equivalence and two critical branchings. By choosing the QNF labelling on Λ induced by the deglex order $y > x$, we get the following Squier's decreasing completion for Λ



By Theorem 4.5, this completion yields a coherent presentation of the algebra $\mathbf{A}$.

References

[1] Clément Alleaume. Linear polygraphs applied to categorification. *ArXiv: 1704.02623*, 2017.

[2] Clément Alleaume. Rewriting in higher dimensional linear categories and application to the affine oriented Brauer category. *Journal of Pure and Applied Algebra*, pages –, 2017.

[3] Clément Alleaume and Philippe Malbos. Coherence of quasi-terminating decreasing 2-polygraphs. *Proceedings of International Workwhop on Confluence*, 2016.

[4] Clément Alleaume and Philippe Malbos. Coherence of string rewriting systems by decreasingness. *ArXiv: 1612.09193*, 2016.

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

[6] Yves Guiraud, Eric Hoffbeck, and Philippe Malbos. Linear polygraphs and Koszulity of algebras. 2014.

[7] Yves Guiraud and Philippe Malbos. Coherence in monoidal track categories. *Mathematical Structures in Computer Science*, 22(6):931–969, 2012.

[8] Nohra Hage and Philippe Malbos. Knuth's coherent presentations of plactic monoids of type A. *Algebras and Representation Theory*, May 2017.

[9] Nagayoshi Iwahori. On the structure of a Hecke ring of a Chevalley group over a finite field. *J. Fac. Sci. Univ. Tokyo Sect. I*, 10:215–236 (1964), 1964.

[10] Deepak Kapur and Paliath Narendran. A finite Thue system with decidable word problem and without equivalent finite canonical system. *Theoret. Comput. Sci.*, 35(2-3):337–344, 1985.

[11] Teo Mora. An introduction to commutative and noncommutative Gröbner bases. *Theoret. Comput. Sci.*, 134(1):131–173, 1994. Second International Colloquium on Words, Languages and Combinatorics (Kyoto, 1992).

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

[13] Vincent van Oostrom. Confluence by decreasing diagrams. *Theoret. Comput. Sci.*, 126(2):259–280, 1994.

Critical Peaks Redefined
$\Phi \sqcup \Psi = \top$

Nao Hirokawa[1], Julian Nagele[2], Vincent van Oostrom[2], and Michio Oyamaguchi[3]

[1] School of Information Science, JAIST, Japan
hirokawa@jaist.ac.jp
[2] Department of Computer Science, University of Innsbruck, Austria
julian.nagele@uibk.ac.at, Vincent.van-Oostrom@uibk.ac.at
[3] Nagoya University, Japan
oyamaguchi@za.ztv.ne.jp

Abstract

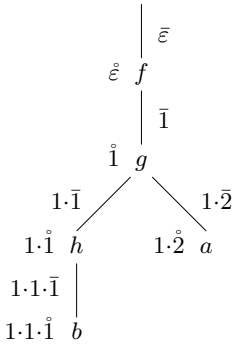
Let a *cluster* be a term with a number of patterns occurring in it. We give two accounts of clusters, a *geometric* one as sets of (node and edge) positions, and an *inductive* one as pairs of terms with gaps (2nd order variables) and pattern-substitutions for the gaps. We show both notions of cluster and the corresponding refinement/coarsening orders on them, to be isomorphic. This equips clusters with a lattice structure which we lift to (parallel/multi) steps to yield an alternative account of the notion of critical peak.

1 Introduction

The critical pair lemma [3] is the cornerstone for proving confluence of first-order term rewrite systems. It expresses that a term rewrite system is locally confluent if and only if all its critical pairs are joinable. In case the system is moreover terminating this allows to reduce, by Newman's Lemma [4], checking confluence to checking joinability of its critical pairs, which are finitely many in the case of a finite term rewrite system. This forms the basis for Knuth–Bendix completion. The termination condition cannot be omitted without more from the critical pair lemma: On the one hand, a non-terminating TRS may fail to be confluent even in the absence of critical pairs due to non-left-linearity, as established by Klop. On the other hand, a non-terminating left-linear TRS may still fail to be confluent despite that all its critical pairs are joinable. Still, for orthogonal, i.e. left-linear and without critical pairs, TRSs confluence does hold for *geometric* reasons: redex-patterns can be contracted independently of each other, inducing a notion of residual. Starting with Church and Rosser a rich theory of residuals has been developed, but comparatively little attention has been paid to the result that lies at its basis: a strengthening of the critical pair lemma stating that any peak *either* can be decomposed into smaller peaks, *or* is (a variable-instance of) a critical or empty (at least one of the steps is empty) peak. We present such a *critical peak* lemma.

Since both for defining rewriting and for defining critical peaks the notion of *encompassment* is essential, we start off with analysing it. In particular, we call a term with a number of patterns (think of left-hand sides of rules) encompassed by it a *cluster*, and introduce two representations of clusters: a *geometric* one as sets of (node and edge) positions, and an *inductive* one as pairs of terms with gaps (2nd order variables) and pattern-substitutions for the gaps. One can think of these two representations as corresponding to the pictures respectively the formal proof of the critical pair lemma as found in e.g. [1, 5, 6]. Here we give formal accounts of *both* and of the refinement/coarsening order on them, and show them to be isomorphic. This allows one to bridge the gap between the often informal geometric intuition ('proofs by picture') at

B. Accatoli and B. Felgenhauer (eds.); IWC 2017, pp. 33–37



(a) Vertex/edge positions [6]

$\emptyset$	$(f(g(h(b),a)),\ [\![\,]\!]) = \bot$
$\{\mathring{\varepsilon}\}$	$(X(g(h(b),a)),\ [\![X := f(\mathsf{v}_1)]\!])$
$\{\mathring{1}, 1{\cdot}\mathring{1}\}$	$(f(X(Y(b),a)),[\![X, Y := g(\mathsf{v}_1,\mathsf{v}_2), h(\mathsf{v}_1)]\!])$
$\{\mathring{1}, 1{\cdot}\bar{1}, 1{\cdot}\mathring{1}\}$	$(f(X(b,a)),\ \ [\![X := g(h(\mathsf{v}_1),\mathsf{v}_2)]\!])$
$\{\mathring{1}, 1{\cdot}\bar{2}, 1{\cdot}\mathring{2}\}$	$(f(X(h(b))),\ \ [\![X := g(\mathsf{v}_1,a)]\!])$
$\left\{\begin{array}{c}\mathring{\varepsilon}, \bar{1}, \mathring{1}, 1{\cdot}\bar{1}, 1{\cdot}\mathring{1},\\ 1{\cdot}1{\cdot}\bar{1}, 1{\cdot}1{\cdot}\mathring{1}, 1{\cdot}\bar{2}, 1{\cdot}\mathring{2}\end{array}\right\}$	$(X,\ \ [\![X := f(g(h(b),a))]\!]) = \top$

(b) Geometric and inductive clusters.

Figure 1: Positions and clusters for $f(g(h(b),a))$.

the basis of properties of residuals, and the inductive nature of ('terms and steps') of term rewriting. As a first example (we anticipate many more) we redefine in this paper the notion of critical peak in a purely lattice theoretic way, based just on the coarsening/refinement order of clusters. More precisely, we call a local peak between steps ϕ and ψ *critical* if $\phi \sqcup \psi = \top$, or in words, the redex-patterns of ϕ and ψ must be overlapping (if not, $\phi \sqcup \psi \sqsubset \top$ as the join would 'miss an edge' and comprise *two* patterns) and all symbols in the source of the peak must be part of either pattern (if not, $\phi \sqcup \psi \sqsubset \top$ as the join would 'miss a node' and not comprise *all* symbols). This definition captures the intuition behind something being *critical*: the source does encompass both redex-patterns but only just so, nothing else is encompassed.

We restrict ourselves to first-order term rewriting and to linear patterns.

2 Clusters and the refinement lattice

We introduce clusters and the refinement order on them. It is standard to represent terms as labelled trees using sequences of positive natural numbers called *positions* for the nodes of the tree. Following [6, Chapter 8] we extend this by having both *vertex* and *edge* positions, see Figure 1(a). Note that $\mathring{\imath}$ and $\bar{\imath}$ stand for $i{\cdot}\mathring{\varepsilon}$ and $i{\cdot}\bar{\varepsilon}$, respectively. Let $i{\cdot}P$ denote $\{i{\cdot}p \mid p \in P\}$.

Definition 1. *The* Tree *algebra has as carrier sets of positions, and interpretations*

$$f^{\mathsf{Tree}}(\vec{P}) = \{\bar{\varepsilon}, \mathring{\varepsilon}\} \cup \bigcup_i i{\cdot}P_i$$

The set of all positions of a term arises by assigning $\{\bar{\varepsilon}, \mathring{\varepsilon}\}$ to variables. We are interested in the *internal* positions arising from assigning $\emptyset$ to variables and removing the root edge $\bar{\varepsilon}$.

Definition 2. *A* cluster *for a given term t is a set of internal positions of t such that if an edge $p{\cdot}\bar{\imath}$ is in the set, its endpoints $p{\cdot}\mathring{\varepsilon}$ and $p{\cdot}\mathring{\imath}$ are too. Its connected components are called* patterns.

Our patterns correspond to those in [6, Chapter 8].

Example 1. *The first column of Figure 1(b) lists some clusters for the term $f(g(h(b),a))$, of which the first and third are not patterns (for the latter, $1{\cdot}\bar{1}$ is missing). Since $\{\mathring{1}, 1{\cdot}\bar{1}\}$ lacks the endpoint $1{\cdot}\mathring{1}$ of the edge $1{\cdot}\bar{1}$, it is not a cluster. Note that $\{\mathring{1}, 1{\cdot}\bar{1}, 1{\cdot}\mathring{1}\} \neq \{\mathring{1}, 1{\cdot}\mathring{1}\}$; the former is a cluster comprising a single pattern, whereas the latter comprises two patterns.*

Lemma 1. *For any term, the clusters for that term constitute a finite distributive lattice with respect to the subset order $\subseteq$.*

Proof. By terms being finite, and properties being inherited from the subset order. □

We give an alternative definition of clusters. To keep both apart, we will refer to the above notions as *geometric* and to the ones introduced below as *inductive*.

Definition 3. *A* skeleton *is constructed from function symbols and* 1*st and* 2*nd order variables, the latter called* gaps*. It is a* pattern*-skeleton of* arity *n, if it is not a variable and* standard*: the vector of variables occurring from left to right is $\mathsf{v}_1, \ldots, \mathsf{v}_n$. A* term *is a skeleton without gaps, and a* pattern *is a pattern-skeleton without gaps. A* cluster *is a pair $(M,[\![\vec{X} := \vec{\ell}]\!])$ with M a skeleton linear in the gaps, and $[\![\vec{X} := \vec{\ell}]\!]$ substituting patterns $\vec{\ell}$ for those, respecting arities. We say $(M,[\![\vec{X} := \vec{\ell}]\!])$ is a cluster* for *the term $M^{[\![\vec{X}:=\vec{\ell}]\!]}$.*

Example 2. *The inductive clusters corresponding to the six geometric clusters of Example 1 are listed in the second column of Figure 1(b). The inequality in that example corresponds to the inequality $(f(X(b,a)),[\![X := g(h(\mathsf{v}_1),\mathsf{v}_2)]\!]) \neq (f(X(Y(b),a)),[\![X,Y := g(\mathsf{v}_1,\mathsf{v}_2),h(\mathsf{v}_1)]\!])$.*

Definition 4. *The* coarsening *order $\sqsubseteq$ (the* refinement *order $\sqsupseteq$) on inductive clusters is defined by $(N,\beta) \sqsupseteq (M,\alpha)$ if $N^\gamma = M$ and $\beta = \alpha \circ \gamma$ for some pattern-skeleton substitution γ.*

Example 3. *For the term $f(a)$ the pattern comprising both symbols may be refined to the cluster comprising two patterns: $(Z,[\![Z := f(a)]\!]) \sqsupset (X(Y),[\![X,Y := f(\mathsf{v}_1),a]\!])$ witnessed by the pattern-skeleton substitution $[\![Z := X(Y)]\!]$. Geometrically this corresponds to $\{\mathring{\varepsilon}, \bar{1}, \mathring{1}\} \supset \{\mathring{\varepsilon}, \mathring{1}\}$.*

The main result of this section can be viewed as an instance of Birkhoff's Fundamental Theorem of Finite Distributive Lattices, expressing that *all* such lattices can be represented via downward-closed sets of join-irreducible elements ordered by subset.

Theorem 1. *For a given term, geometric clusters ordered by $\subseteq$ are isomorphic to inductive clusters, up to renaming of gaps, ordered by $\sqsubseteq$. The order is a finite distributive lattice.*

Proof. First note that we can map any inductive cluster $(M,[\![\vec{X} := \vec{\ell}]\!])$ to a geometric cluster by means of what we call a *cluster* algebra, a pair of algebras for interpreting both components:

- Shift for interpreting M: $f^{\mathsf{Shift}}(\vec{P}) = \bigcup_i i{\cdot}P_i$; and
- Tree for interpreting $\vec{X}$ via $\vec{\ell}$: $f^{\mathsf{Tree}}(\vec{P}) = \{\bar{\varepsilon}, \mathring{\varepsilon}\} \cup f^{\mathsf{Shift}}(\vec{P})$, then removing the root edge.

This map is seen to be a bijection. That it preserves the order is seen:

(geometric ⇒ inductive) by induction on the term, simultaneously building the inductive clusters from both geometric clusters and the witnessing pattern-skeleton substitution, using that geometric clusters are preserved under left-quotienting by argument positions.

(inductive ⇒ geometric) algebraically, using a substitution lemma and that the Tree-interpretation contains the Shift-interpretation; □

Example 4. $(f(X(b,a)),[\![X := g(h(\mathsf{v}_1),\mathsf{v}_2)]\!])^{(\mathsf{Shift},\mathsf{Tree})} = 1{\cdot}(g(h(\mathsf{v}_1),\mathsf{v}_2)^{\mathsf{Tree}}_{[\mathsf{v}_1,\mathsf{v}_2\mapsto\emptyset,\emptyset]} - \{\bar{\varepsilon}\}) = 1{\cdot}(\{\bar{\varepsilon},\mathring{\varepsilon},\bar{1},\mathring{1}\} - \{\bar{\varepsilon}\}) = 1{\cdot}\{\mathring{\varepsilon},\bar{1},\mathring{1}\} = \{\mathring{1}, 1{\cdot}\bar{1}, 1{\cdot}\mathring{1}\}$.

Equipping clusters with the lattice operators $\top$, $\bot$, $\sqcap$, $\sqcup$, the theorem shows the join-irreducible elements of the refinement order can be perceived as vertices (patterns comprising a single function symbol) and edges (patterns comprising two function symbols), which can be seen as justifying having both types of positions: an edge is *more* than the join of its endpoints.

3 Critical peaks redefined

We first redefine single/parallel/multi-steps in first-order term rewriting *by second-order means* as clusters with rule symbols [6] in patterns, and next critical peaks via the clusters of its steps.

Lemma 2. *For a left-linear TRS, $t \to s$ iff $t = M^{[\![X:=\ell]\!]}$ and $M^{[\![X:=r]\!]} = s$ for some skeleton M and pattern substitution $[\![X := \ell]\!]$, for rule $\ell \to r$ with ℓ standard.*

Proof. If $t = C[\ell^\sigma]$ and $C[r^\sigma] = s$, then set $M = C[X(\vec{\mathsf{v}}^\sigma)]$ and vice versa. □

Turning rules into rule symbols, the lemma justifies representing a step $t \to s$ as a cluster ϕ having the rule symbol as pattern substitution. We denote it by $t \to_\phi s$.

Example 5. *The step $f(f(f(a))) \to f(g(f(a), f(a)))$ for the rule $\varrho(\mathsf{v}_1) : f(\mathsf{v}_1) \to g(\mathsf{v}_1, \mathsf{v}_1)$ can be represented by the cluster $(f(X(f(a))), [\![X := \varrho(\mathsf{v}_1)]\!])$: projecting the rule ϱ in the substitution to its left/right-hand side $[\![X := f(\mathsf{v}_1)]\!]/\,[\![X := g(\mathsf{v}_1, \mathsf{v}_1)]\!]$ yields the step.*

We now use the lattice to measure the interaction between steps in peaks.[1] Note that the top element $\top$ for an n-ary pattern ℓ has shape $(X(\mathsf{v}_1, \ldots, \mathsf{v}_n), [\![X := \ell]\!])$.

Definition 5. *A local peak $s \,{}_\phi\!\leftarrow t \to_\psi u$ is* critical *if $\phi \sqcup \psi = \top$ with t standard, where we extend the refinement order to steps via their left-hand side.*

Example 6. *Consider the (standard) rules $f(\mathsf{v}_1) \to \mathsf{v}_1$ and $f(\mathsf{v}_1) \to a$.*

- *$\mathsf{v}_1 \leftarrow f(\mathsf{v}_1) \to a$ is critical since the union of the redex-patterns is $\{\mathring{\varepsilon}\}$, the set of all internal positions of $f(\mathsf{v}_1)$;*
- *$b \leftarrow f(b) \to a$ is not critical since the union of the redex-patterns is $\{\mathring{\varepsilon}\}$, which is distinct from the set $\{\mathring{\varepsilon}, \bar{1}, \mathring{1}\}$ of all internal positions of $f(b)$; and*
- *$a \leftarrow f(f(\mathsf{v}_1)) \to f(a)$ is not critical since the union of the redex-patterns $\{\mathring{\varepsilon}, \mathring{1}\}$ misses the internal position $\bar{1}$ of $f(f(\mathsf{v}_1))$.*

Lemma 3. *The definition of critical peak for a pair of rules is equivalent to the definitions found in the literature, up to* most generalness *(unifier or common instance),* chiasmus *(1st rule–2nd rule vs. 2nd rule–1st rule),* order *(outer–inner vs. inner–outer),* renaming *(variables in the peak), and* triviality *(overlap of a rule with itself at the root).*

Proof. The definition of critical pair/peak varies along these parameters throughout the standard literature [3, 2, 1, 5, 6]. The notions in the literature *implement* our abstract notion. □

The above generalises to multi-steps [6] contracting a number of (non-overlapping) redex-patterns at the same time. We write $\multimap\!\!\to$ ($\multimap\!\!\to_\Phi$) for multi-step (induced by cluster Φ).

Lemma 4. *For a left-linear TRS, $t \multimap\!\!\to s$ iff $t = M^{[\![\vec{X}:=\vec{\ell}]\!]}$ and $M^{[\![\vec{X}:=\vec{r}]\!]} = s$, for some skeleton M and pattern substitution $[\![\vec{X} := \vec{\ell}]\!]$, for rules $\overrightarrow{\ell \to r}$ with $\vec{\ell}$ standard.*

Definition 6. *A peak $s \,{}_\Phi\!\leftarrow\!\!\multimap t \multimap\!\!\to_\Psi u$ is* critical *if $\Phi \sqcup \Psi = \top$ with t standard and $\Phi, \Psi \neq \bot$, where we extend the refinement order to multi-steps via their left-hand side.*

That is, the same concise (5 symbols) definition of critical peak as before allows us to capture all the notions of parallel critical peaks and development critical peaks (having definitions of up to 2 pages), due to Gramlich, Toyama, Okui, and Felgenhauer from the literature.

[1] The lattice structure on clusters does, in itself, not give rise to a lattice structure on rules/steps/reductions.

Example 7. *The following are critical peaks in our sense:*

- *the parallel (one–parallel) critical peak* $c \leftarrow f(a,a) \twoheadrightarrow f(b,b)$ *for rules* $f(a,a) \to c$, $a \to b$;
- *the development (one–multi) critical peak* $g(c) \leftarrow g(f(a)) \multimap b$ *for rules* $f(a) \to c$, $g(f(\mathsf{v}_1)) \to \mathsf{v}_1$, $a \to b$; *and*
- *the multi–multi critical peaks* $f(g^n(\mathsf{v}_1)) \leftarrow\!\!\circ\; f^{2n+1}(\mathsf{v}_1) \multimap g^n(f(\mathsf{v}_1))$ *for* $f(f(\mathsf{v}_1)) \to g(\mathsf{v}_1)$.

Lemma 5 (Critical peak)**.** *If* $s \;{}_{\Phi}\!\leftarrow\!\!\circ\; t \multimap_{\Psi} u$, *then the size of the skeleton of* $\Phi \sqcup \Psi$ *is* either

- $\leqslant 1$*: the peak is a variable-substitution instance of either a* critical *(*$\Phi \sqcup \Psi = \top$ *and* $\Phi, \Psi \neq \bot$*) or an* empty *(if* $\Phi \neq \bot$ *then* $\Psi = \bot$ *(note* $\Phi = \top$*), and vice versa) peak;* or
- > 1*:* $\Phi = \Phi_0^{[x:=\Phi_1]}$ *and* $\Psi = \Psi_0^{[x:=\Psi_1]}$*, for peaks* $s_i \;{}_{\Phi_i}\!\leftarrow\!\!\circ\; t_i \multimap_{\Psi_i} u_i$ *with* $i \in \{0,1\}$*, having skeletons of* smaller *sizes.*

We sketch how the lemma allows to prove many confluence-by-critical-pair-analysis results by induction on the size of the skeleton and splitting in these two cases.

Example 8.

- *the critical pair lemma for (left-linear) TRSs follows in the first case by the assumption that critical pairs are joinable, and in the second case by the induction hypothesis (twice) and then using that reduction is closed under substitution to* recompose*;*
- *that orthogonal TRSs are confluent follows by proving that multi-steps have the diamond property, with the first case being trivial since each critical peak is trivial by orthogonality, and concluding in the second case by the induction hypothesis (twice) and then using that multi-steps are closed under substitution to* recompose *the multi-steps; and*
- *that development-closed TRSs are confluent is proven by refining the proof of the previous item by an extra (outer) induction on the* amount *of overlap between the multi-steps. For that it is essential that the refinement order is a* distributive *lattice, as that allows to express the amount of overlap between two multi-steps as the sum of the amounts of overlap (the sizes of the meets) between their constituting redex-patterns.*

We expect the above extends to non-left-linear,[2] higher-order pattern, and graph rewriting.

Acknowledgements. This work has been partially supported by Austrian Science Fund (FWF) project P27528, JSPS KAKENHI Grant Number 17K00011, and JSPS Core to Core Program. We thank A. Middeldorp, B. Felgenhauer, and participants of TeReSe (Eindhoven 2017) and the Innsbruck master seminar for discussions, and the IWC reviewers for feedback.

References

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

[2] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of TCS*, volume B, Formal Models and Semantics, pages 243–320. Elsevier, 1990.

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

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

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

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

[2] Although the definition of critical peak remains the same, we then do not get a distributive lattice.

CoCoWeb

A Convenient Web Interface for Confluence Tools*

Julian Nagele and Aart Middeldorp

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

Abstract

We present a useful web interface for tools that participate in the annual confluence competition.

1 Introduction

In recent years several tools have been developed to automatically prove confluence and related properties of a variety of rewrite formats. These tools compete annually in the confluence competition [1] (CoCo).[1] Most of the tools can be downloaded, installed, and run on one's local machine, but this can be a painful process.[2] Few confluence tools—we are aware of CO3 [4], ConCon [5], and CSI [3,7]—provide a convenient web interface to painlessly test the status of a system that is provided by the user.

Inspired by the latest web interface of CSI [3], in this note we present CoCoWeb, a web interface that provides a single entry point to all tools that participate in CoCo. CoCoWeb is available at

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

The typical use of CoCoWeb will be to test whether a given confluence problem is known to be confluent or not. This is useful when preparing or reviewing an article, preparing or correcting exams about term rewriting, and when contemplating to submit a challenging problem to the confluence problems (Cops)[3] database. In particular, CoCoWeb is useful is when looking for (killer) examples to illustrate a new technique. For instance, in [2] a rewrite system is presented that can be shown to be confluent with the technique introduced in that paper. The authors write "Note that we have tried to show confluence [...] by confluence checker ACP and Saigawa, and both of them failed." Despite having an easy to use web interface, CSI was not tried. CoCoWeb could also be useful for the CoCo steering committee when integrating newly submitted problems into Cops.

The tools run on the same hardware, which is compatible with a single node of StarExec [6] that is used for CoCo, allowing for a proper comparison of tools.

In the next section we present the web interface of CoCoWeb by means of a number of screenshots. Implementation details are presented in Section 3 and in Section 4 we list some possibilities for extending the functionality of CoCoWeb in the future.

*This work is supported by the Austrian Science Fund (FWF): project P27528.

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

[2] StarExec provides a VM Image with their environment, which can helpful in case a local setup is essential.

[3] http://cops.uibk.ac.at

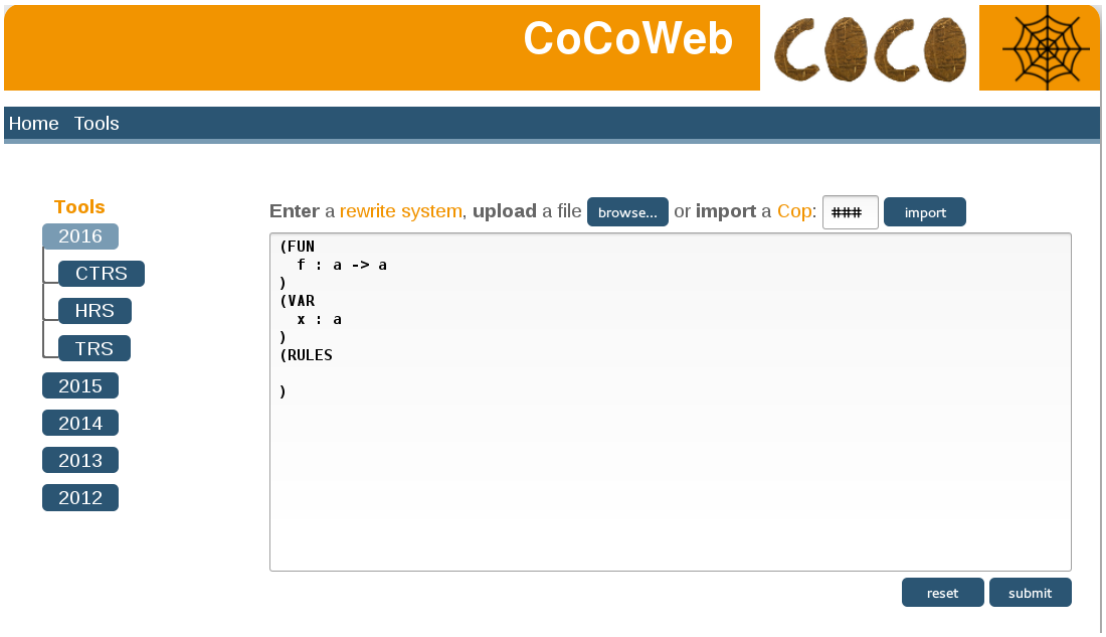


Figure 1: The entry page of CoCoWeb.

2 Web Interface

Figure 1 shows a screenshot of the entry page of CoCoWeb. Problems can be entered in three different ways: (1) using the text box, (2) uploading a file, (3) entering the number of a system in Cops. The tools that should be executed can be selected from the tools panel on the left. Tools are grouped into categories, similar to the grouping in CoCo except that we merged the

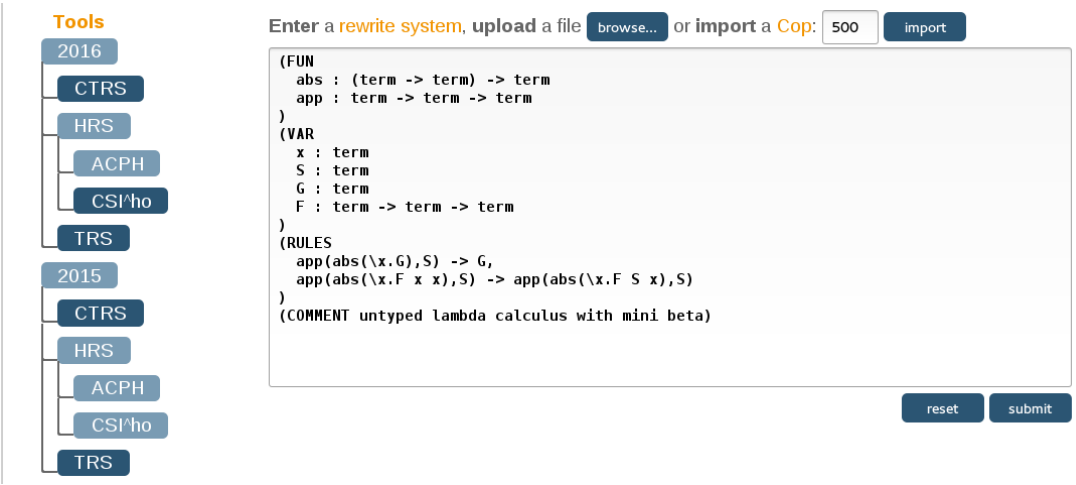


Figure 2: Problem and tool selection in CoCoWeb.

certified categories with the corresponding uncertified categories. Multiple tools can be selected. This is illustrated in Figure 2. Here we selected the CoCo 2016 and 2015 versions of ACPH and the CoCo 2015 version of CSIˆho, and Cop 500 is chosen is input problem.

The screenshot in Figure 3 shows the output of CoCoWeb after clicking the submit button. The output of the selected tools is presented in separate tabs. The colors of these tabs reveal useful information: Green means that the tool answered yes, red (not shown) means that the tool answered no, and a maybe answer or a timeout is shown in blue. By clicking on a tab, the color is made lighter and the output of the tool is presented. The final line of the output is timing information provided by CoCoWeb.

The final screenshot (in Figure 4) is concerned with the example in [2] that we referred to in the introduction.

3 Implementation

Most of CoCoWeb is built using PHP. User input in forms, i.e., rewrite systems and tool selections, is sent using the HTTP POST method. The dynamic parts of the website, namely folding and unfolding in the tool selection menu and the tabs used for viewing tool output are implemented using JavaScript.

To layout the tool selection menu we made extensive use of CSS3 selectors. For instance, the buttons to select tools are implemented as checkboxes with labels that are styled according to whether the checkbox is ticked or not:

```
.tools input[type="checkbox"]:checked + label {
  color: white;
  background-color: #799BB3;
}
```

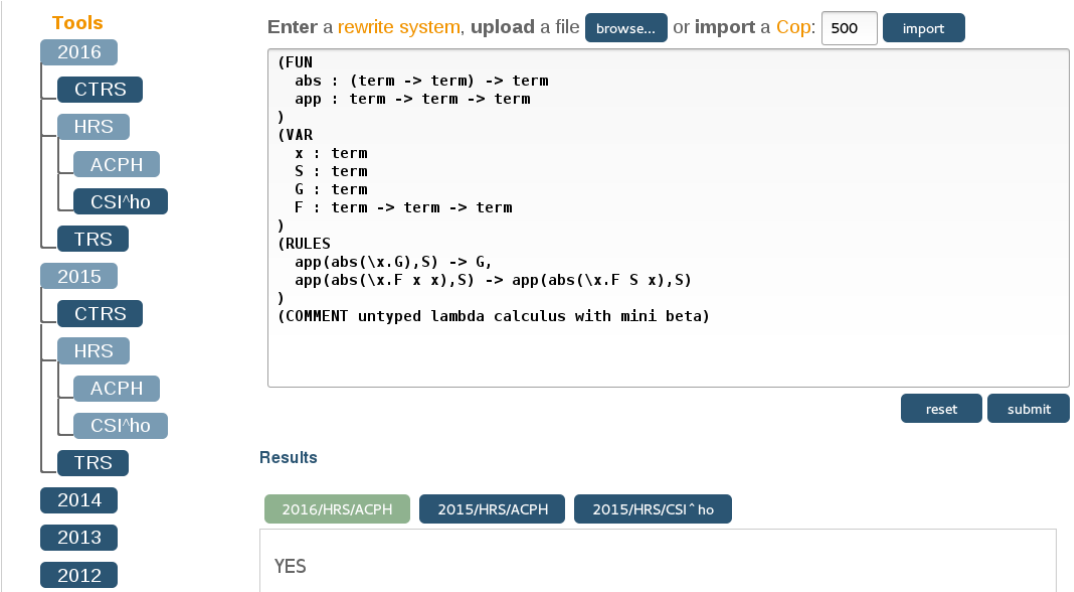


Figure 3: Result displaying in CoCoWeb.

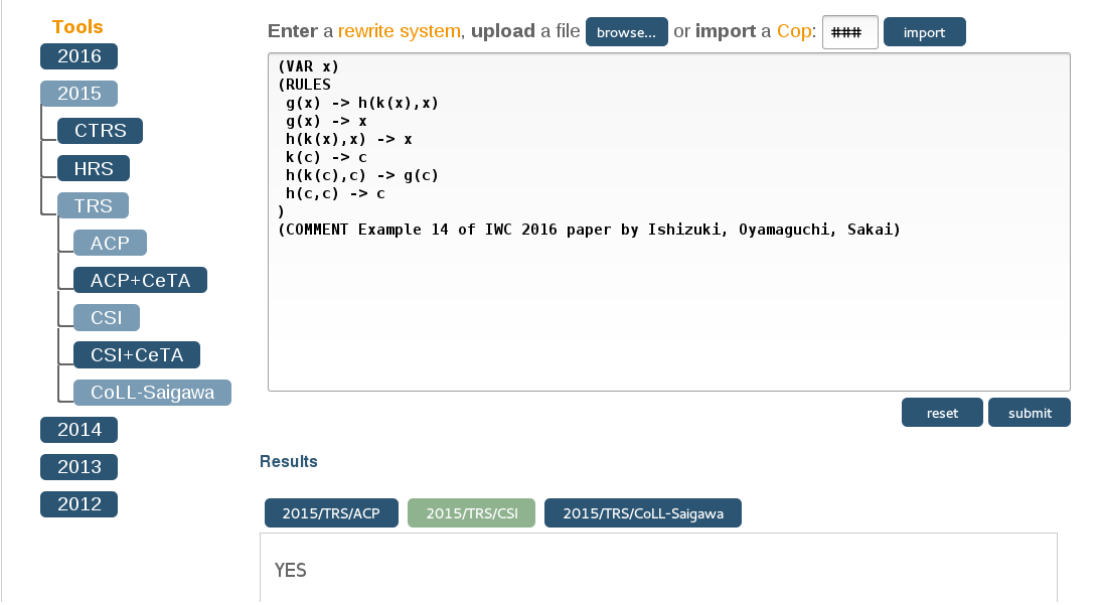


Figure 4: Testing Example 14 from [2] in CoCoWeb.

Drawing the edges of the tree menu is also done using CSS, relying mainly on its `::before` selector.

The content of the tool menu, i.e., years, the grouping by categories, and the actual tools, is generated automatically from a directory tree that has the structure of the menu in CoCoWeb. There small configuration files reside that specify how the tools are to be run, in case they are selected. Two environment variables are set in such a file, for example the one for the 2012 version of Saigawa reads as follows:

```
TOOLDIR="Saigawa-2012/bin"
TOOL="./starexec_run_saigawa -t $TO $FILE"
```

The variable `TOOLDIR` specifies the directory that contains the tool binary, while `TOOL` gives the tool invocation, which in turn refers to `TO`, the timeout, and `FILE`, the input rewrite system. Using such configuration files tools are run using the following script, whose first and second argument are the configuration of the tool and input rewrite system respectively:

```
DIR=$(pwd -P)
FILE=$(readlink -f $2)
TO=59
TOT=61
TOK=63
source $1
pushd $DIR/bin/$TOOLDIR > /dev/null
/usr/bin/time -f "\nTook %es" timeout -k $TOK $TOT $TOOL
popd > /dev/null
```

The script uses three different timeouts: `TO` is the timeout passed to the tool itself if supported, while after `TOT` and `TOK` the signals `SIGTERM` and `SIGKILL` are sent to the tool in case it did

not terminate on its own volition. When multiple tools are selected, CoCoWeb runs them sequentially, in order to avoid interference.

4 Possible Extensions

We conclude this note with some ideas for future extensions of the functionality of CoCoWeb.

- By analyzing the format of the input problem, it is often possible to determine the category to which the problem belongs. This information can then be used to restrict tool selection accordingly.
- Adding support for other competition categories like ground confluence and unique normal forms is an obvious extension. This, however, limits the possibilities to restrict tool selection mentioned in the previous item.
- Selected tools are executed sequentially and so if many tools are selected, the 60 seconds time limit per tool may result in an unacceptably slow response. In that case a timer option is a welcome feature.

References

[1] T. Aoto, N. Hirokawa, J. Nagele, N. Nishida, and H. Zankl. Confluence Competition 2015. In *Proc. 25th International Conference on Automated Deduction*, volume 9195 of *Lecture Notes in Artificial Intelligence*, pages 101–104, 2015. doi: 10.1007/978-3-319-21401-6_5.

[2] S. Ishizuki, M. Oyamaguchi, and M. Sakai. Conditions for confluence of innermost terminating term rewriting systems. In *Proc. 5th International Workshop on Confluence*, pages 70–74, 2016. Available from http://cl-informatik.uibk.ac.at/iwc/iwc2016.pdf.

[3] J. Nagele, B. Felgenhauer, and A. Middeldorp. CSI: New evidence – a progress report. In *Proc. 26th International Conference on Automated Deduction*, volume 10395 of *Lecture Notes in Artificial Intelligence*, pages 385–397, 2017. doi: 10.1007/978-3-319-63046-5_24.

[4] N. Nishida, T. Kuroda, M. Yanagisawa, and K. Gmeiner. CO3: A COnverter for proving COnfluence of COnditional TRSs (version 1.2). In *Proc. 4th International Workshop on Confluence*, page 42, 2015. Available from http://cl-informatik.uibk.ac.at/iwc/iwc2015.pdf.

[5] T. Sternagel and A. Middeldorp. Conditional confluence (system description). In *Proc. Joint 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications*, volume 8560 of *Lecture Notes in Computer Science (Advanced Research in Computing and Software Science)*, pages 456–465, 2014. doi: 10.1007/978-3-319-08918-8_31.

[6] A. Stump, G. Sutcliffe, and C. Tinelli. StarExec: A cross-community infrastructure for logic solving. In *Proc. 7th International Joint Conference on Automated Reasoning*, volume 8562 of *Lecture Notes in Artificial Intelligence*, pages 367–373, 2014. doi: 10.1007/978-3-319-08587-6_28.

[7] H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – A confluence tool. In *Proc. 23rd International Conference on Automated Deduction*, volume 6803 of *Lecture Notes in Artificial Intelligence*, pages 499–505, 2011. doi: 10.1007/978-3-642-22438-6_38.

A Ground Joinability Criterion for Ordered Completion*

Sarah Winkler

University of Innsbruck, Austria
sarah.winkler@uibk.ac.at

Abstract

By Newman's Lemma and the Extended Critical Pair Lemma, ordered completion tools can establish ground confluence by ensuring ground joinability of all extended critical pairs. We present a new criterion for ground joinability over a given signature that extends the test by Martin and Nipkow (1990). The criterion is particularly useful when a suitable reduction order is to be discovered by means of a SAT/SMT-based search.

1 Introduction

Ordered completion [3] has been a highly influential calculus in automated deduction as it is refutationally complete for equational theorem proving. If no goal is supplied or the goal cannot be proven, ordered completion attempts to derive a presentation of the equational theory that is ground complete, i.e., terminating and ground confluent.

In theorem proving, ground completeness was mostly considered over a signature $\mathcal{F}^{\mathcal{C}}$ that extends the signature $\mathcal{F}$ of the (finite) input problem by infinitely many constants. Naturally ground completeness over $\mathcal{F}^{\mathcal{C}}$ implies ground completeness over $\mathcal{F}$, but the following example shows that the reverse does not hold.

Example 1. The rewrite system $\mathcal{R}$ consisting of the three rules

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

is terminating and ground confluent over the signature $\mathcal{F} = \{\mathsf{f}, \mathsf{g}, \mathsf{a}\}$ as every ground term rewrites to a. But in presence of an additional constant c there is the non-joinable peak $\mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{c}))))) \leftarrow \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{c})))))) \to \mathsf{f}(\mathsf{g}(\mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{c}))))))$, so $\mathcal{R}$ is not ground confluent over $\mathcal{F}^{\mathcal{C}}$.

It is actually often sufficient if a system is ground complete for a fixed, finite signature $\mathcal{F}$, e.g., to witness satisfiability of a goal. However, this comes at the price of a profound difference with respect to computability: while ground confluence of ordered rewriting is decidable over $\mathcal{F}^{\mathcal{C}}$ for a large class of reduction orders [4] it is undecidable over $\mathcal{F}$, despite termination [5].

By Newman's Lemma and the Extended Critical Pair Lemma [3] ground confluence of a terminating system can be established by ensuring that all extended critical pairs are ground joinable. In this paper we propose a new ground joinability criterion for this setting that extends the test proposed by Martin and Nipkow [8]. It is strictly more powerful as far as ground completeness over the original signature $\mathcal{F}$ is concerned. In particular, our criterion produces constraints on the reduction order that are necessary for ground joinability. Using the fact that satisfiability of LPO and KBO constraints is in NP [9, 7], we exploit this test in the ordered completion tool MædMax, where maxSAT/maxSMT guides the search for a reduction order that admits a ground-complete presentation.

The remainder of this paper is organized as follows. In Section 2 we recall some relevant notions. In Section 3 the ground joinability criterion is explained. In Section 4 we add some remarks on our implementation and conclude.

*This work is supported by FWF (Austrian Science Fund) project T789.

2 Preliminaries

In the sequel standard no(ta)tions from term rewriting are used [2]. We consider the set of (ground) terms $\mathcal{T}(\mathcal{F},\mathcal{V})$ ($\mathcal{T}(\mathcal{F})$) over a finite signature $\mathcal{F}$ that is assumed to contain a constant. The signature $\mathcal{F}^{\mathcal{C}}$ extends $\mathcal{F}$ by an infinite set of constants $\mathcal{C}$. A substitution σ is *$\mathcal{F}$-grounding* for a set of variables V if $\sigma(x) \in \mathcal{T}(\mathcal{F})$ for all $x \in V$, and $\mathcal{F}$-grounding for a term t if it is $\mathcal{F}$-grounding for $\mathcal{V}\mathrm{ar}(t)$. Given a TRS $\mathcal{R}$, terms s and t are *ground joinable on* $\mathcal{F}$, denoted $s \Downarrow_{\mathcal{F}} t$, if $s\sigma \downarrow_{\mathcal{R}} t\sigma$ holds for all $\mathcal{F}$-grounding substitutions σ. A substitution σ is $\mathcal{R}$-reducible if $\sigma(x)$ is $\mathcal{R}$-reducible for some $x \in \mathcal{D}\mathrm{om}(\sigma)$.

We consider simplification orders $>$ that are ground total. (Any ground-total reduction order can be extended to enjoy the subterm property [2]). These properties are for instance satisfied by lexicographic path orders (LPO) and Knuth-Bendix orders (KBO) with a total precedence [2]. *Ordered rewriting* is a key notion for ordered completion: For a set of equations $\mathcal{E}$ and a reduction order $>$, the (infinite) TRS $\mathcal{E}^{>}$ consists of all rewrite rules $\ell\sigma \to r\sigma$ such that $\ell \approx r \in \mathcal{E} \cup \mathcal{E}^{-1}$ and $\ell\sigma > r\sigma$. A TRS $\mathcal{R}$ is *ground confluent* with respect to $\mathcal{F}$ if $s \,{}^{*}_{\mathcal{R}}\!\leftarrow \cdot \to^{*}_{\mathcal{R}} t$ implies $s \to^{*}_{\mathcal{R}} \cdot \,{}^{*}_{\mathcal{R}}\!\leftarrow t$ for all $s,t \in \mathcal{T}(\mathcal{F})$. A terminating TRS $\mathcal{R}$ is ground confluent if the set of extended critical pairs $\mathrm{CP}_{>}(\mathcal{R})$ are ground joinable [3].

3 A Criterion for Ground Joinability

We build on the idea by Martin and Nipkow [8] to perform a case distinction by considering ordered rewriting using all extensions of $>$ that reflect possible combinations of $\sigma(x) > \sigma(y)$, $\sigma(y) > \sigma(x)$ or $\sigma(x) = \sigma(y)$ for $x, y \in \mathcal{V}$ and a grounding substitution σ.

Suppose $\mathcal{F}$ contains the signature of the input problem, and $>$ is an $\mathcal{F}$-ground-total simplification order. Let O be a set of *ordering constraints*, i.e., a set of variable pairs $x >_{\mathsf{O}} y$ or $x =_{\mathsf{O}} y$. Let $\succeq_{\mathsf{O}}$ be a quasi-order on $\mathcal{T}(\mathcal{F},\mathcal{V})$ that contains $>$ and O, and $\succ_{\mathsf{O}}$ be its strict part. For a substitution σ that is $\mathcal{F}$-grounding on $\mathcal{V}\mathrm{ar}(\mathsf{O})$ and a set of ordering constraints O, the quasi-order $\succeq_{\mathsf{O}}$ *covers* σ if $\sigma(x) > \sigma(y)$ ($\sigma(x) = \sigma(y)$) holds for all $x >_{\mathsf{O}} y$ ($x =_{\mathsf{O}} y$) in O. Moreover, $\succ_{\mathsf{O}}$ is *compatible* with $>$ if $s \succ_{\mathsf{O}} t$ implies $s\sigma > t\sigma$ for all $s,t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ and all substitutions σ that are $\mathcal{F}$-grounding for s and t and covered by $\succeq_{\mathsf{O}}$.

For instance, if $>$ is an LPO and $x >_{\mathsf{O}} y \in \mathsf{O}$ then $\succeq_{\mathsf{O}}$ covers $\sigma = \{x \mapsto \mathsf{g}(\mathsf{a}), y \mapsto \mathsf{a}\}$. If $\succ_{\mathsf{O}}$ is compatible with $>$ then $\mathsf{f}(\mathsf{f}(x,z),y) \succ_{\mathsf{O}} \mathsf{f}(y,\mathsf{f}(x,z))$ may hold, but $\mathsf{f}(y,x) \succ_{\mathsf{O}} \mathsf{f}(x,y)$ cannot.

Lemma 2. *[8, 1] Suppose $\succ_{\mathsf{O}}$ is compatible with $>$ and $\succeq_{\mathsf{O}}$ covers a substitution σ that is $\mathcal{F}$-grounding for s and t. Then $s \to_{\mathcal{E}^{\succ_{\mathsf{O}}}} t$ implies $s\sigma \to_{\mathcal{E}^{>}} t\sigma$.* □

In the remainder of this section we assume that $\succ_{\mathsf{O}}$ is compatible with $>$. Next, we present the inference system GJ which is used to conclude ground joinability of a given equation.

Definition 3. Let $\mathcal{E}$ be a given equational system. The inference system GJ operates on a set $\mathcal{G}$ of tuples $(s \approx t, \mathsf{O}, \sigma)$ where $s \approx t$ is an equation over $\mathcal{T}(\mathcal{F},\mathcal{V})$, O is a set of ordering constraints, and σ is a substitution. It consists of the following seven inference rules:

delete	$\mathcal{G} \uplus \{(s \approx s, \mathsf{O}, \sigma)\} \Rightarrow \mathcal{G}$
rewrite	$\mathcal{G} \uplus \{(s \approx t, \mathsf{O}, \sigma)\} \Rightarrow \mathcal{G} \cup \{(u \approx t, \mathsf{O}, \sigma)\}$
	$\mathcal{G} \uplus \{(t \approx s, \mathsf{O}, \sigma)\} \Rightarrow \mathcal{G} \cup \{(t \approx u, \mathsf{O}, \sigma)\}$
	if $s \to_{\mathcal{E}^{\succ_{\mathsf{O}}}} u$
equation	$\mathcal{G} \uplus \{(s \approx t, \mathsf{O}, \sigma)\} \Rightarrow \mathcal{G}$
	if $s \leftrightarrow_{\mathcal{E}} t$

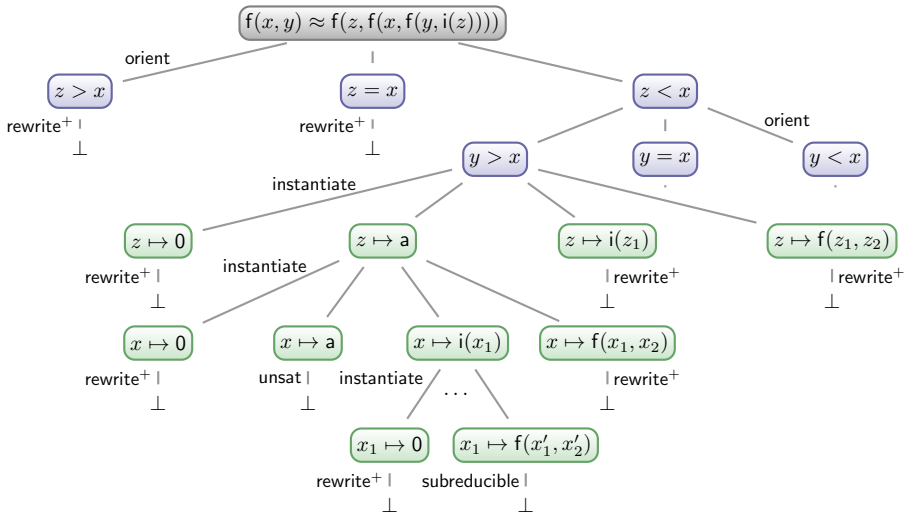


Figure 1: Proving ground joinability of $\mathsf{f}(x,y) \approx \mathsf{f}(z,\mathsf{f}(x,\mathsf{f}(y,\mathsf{i}(z))))$.

orient	$\mathcal{G} \uplus \{(s \approx t, \mathsf{O}, \sigma)\} \Rightarrow \mathcal{G} \cup \{(s \approx t, \mathsf{O} \cup \{x > y\}, \sigma), (s \approx t, \mathsf{O} \cup \{y > x\}, \sigma), (s\rho \approx t\rho, \mathsf{O}\rho, \sigma\rho)\}$
	if $x, y \in \mathcal{V}\mathrm{ar}(s \approx t)$ and where $\rho = \{x \mapsto y\}$
unsat	$\mathcal{G} \uplus \{(s \approx t, \mathsf{O}, \sigma)\} \Rightarrow \mathcal{G}$
	if $\succeq_{\mathsf{O}}$ does not cover any $\mathcal{F}$-grounding substitution for $s \approx t$
instantiate	$\mathcal{G} \uplus \{(s \approx t, \mathsf{O}, \sigma)\} \Rightarrow \mathcal{G} \cup \{(s\sigma_f \approx t\sigma_f, \mathsf{O}\sigma_f, \sigma\sigma_f) \mid f \in \mathcal{F}\}$
	if $\mathcal{F}$ is finite, $\sigma_f = \{x \mapsto f(z_0, \ldots, z_n)\}$ for $x \in \mathcal{V}\mathrm{ar}(s \approx t)$ and $z_0, \ldots, z_n$ fresh
subreducible	$\mathcal{G} \uplus \{(s \approx t, \mathsf{O}, \sigma)\} \Rightarrow \mathcal{G}$
	if σ is $\mathcal{E}^{\succ_{\mathsf{O}}}$-reducible

The inference system GJ extends the transformation relation used by Martin and Nipkow [8] by the last three rules. A sequence of transformation steps $\gamma\colon \mathcal{G}_0 \Rightarrow \mathcal{G}_1 \Rightarrow \mathcal{G}_2 \Rightarrow \ldots$ is called a *derivation*. The aim of this section is the following correctness result:

Lemma 4. *If* $\{(s \approx t, \varnothing, \varnothing)\} \Rightarrow^+ \varnothing$ *then* $s \Downarrow_{\mathcal{F}} t$ *holds.*

Before proving Lemma 4 we illustrate our approach by an example, which also shows that GJ is strictly more powerful than the transformation relation by Martin and Nipkow [8].

Example 5. Consider the following system $\mathcal{E}$ for commutative groups, which is ground complete on both $\mathcal{F} = \{\mathsf{f}, \mathsf{i}, 0\}$ and $\mathcal{F}_{\mathsf{a}} = \mathcal{F} \cup \{\mathsf{a}\}$ [8].

$$\begin{array}{llll}
\mathsf{f}(\mathsf{i}(x), x) \to 0 & \mathsf{f}(y, \mathsf{f}(\mathsf{i}(y), x)) \to x & \mathsf{i}(0) \to 0 & \mathsf{f}(\mathsf{f}(x,y),z) \to \mathsf{f}(x, \mathsf{f}(y,z)) \\
\mathsf{f}(x, \mathsf{i}(x)) \to 0 & \mathsf{f}(\mathsf{i}(y), \mathsf{f}(y, x)) \to x & \mathsf{i}(\mathsf{i}(x)) \to x & \mathsf{i}(\mathsf{f}(x,y)) \to \mathsf{f}(\mathsf{i}(y), \mathsf{i}(x)) \\
\mathsf{f}(x, 0) \to x & \mathsf{f}(\mathsf{i}(y), \mathsf{f}(x, y)) \to x & \mathsf{f}(x,y) \approx \mathsf{f}(y,x) & \mathsf{f}(x, \mathsf{f}(y,z)) \approx \mathsf{f}(y, \mathsf{f}(x,z)) \\
\mathsf{f}(0, x) \to x & \mathsf{f}(y, \mathsf{f}(x, \mathsf{i}(y))) \to x &&
\end{array}$$

Equation $\mathsf{f}(x,y) \approx \mathsf{f}(z,\mathsf{f}(x,\mathsf{f}(y,\mathsf{i}(z))))$ $(\star)$ occurs in the set $\mathrm{CP}_>(\mathcal{E})$ for all reduction orders $>$ that orient the rules as indicated. Figure 1 depicts a possible GJ-derivation for this problem. We cannot rewrite Equation $(\star)$, so we may start by applying the orient rule to x and z. If $z >_\mathsf{O} x$ holds then also $\mathsf{i}(z) >_\mathsf{O} x$ does because $>$ and hence $>_\mathsf{O}$ has the subterm property. It is then easy to check that repeated applications of rewrite turn the problem into a trivial one. But for instance for the variable order $y >_\mathsf{O} x >_\mathsf{O} z$ no rewrite step is possible. We instead continue by applying instantiate to z, writing t for $\mathsf{f}(z,\mathsf{f}(x,\mathsf{f}(y,\mathsf{i}(z))))$:

- If $\sigma_0(z) = \mathsf{0}$ then repeated applications of rewrite turn the problem into a trivial one because of the rewrite steps $t\sigma_0 = \mathsf{f}(\mathsf{0},\mathsf{f}(x,\mathsf{f}(y,\mathsf{i}(\mathsf{0})))) \to \mathsf{f}(x,\mathsf{f}(y,\mathsf{i}(\mathsf{0}))) \to^+ \mathsf{f}(x,y)$.
- If $\sigma_\mathsf{i}(z) = \mathsf{i}(z_1)$ we have $y >_\mathsf{O} x >_\mathsf{O} \mathsf{i}(z_1)$. Since $>$ is a simplification order, $x >_\mathsf{O} \mathsf{i}(z_1)$ implies $x >_\mathsf{O} z_1$ and similar for y. The following rewrite steps make the problem trivial:

$$t\sigma_\mathsf{i} \to \mathsf{f}(\mathsf{i}(z_1),\mathsf{f}(x,\mathsf{f}(y,z_1)))) \to \mathsf{f}(\mathsf{i}(z_1),\mathsf{f}(x,\mathsf{f}(z_1,y)))) \to \mathsf{f}(\mathsf{i}(z_1),\mathsf{f}(z_1,\mathsf{f}(x,y)))) \to \mathsf{f}(x,y)$$

- Finally, if $z = \mathsf{f}(z_1,z_2)$, so $y >_\mathsf{O} x >_\mathsf{O} \mathsf{f}(z_1,z_2)$ then we can first apply rewrite to obtain $\mathsf{f}(\mathsf{f}(z_1,z_2),\mathsf{f}(x,\mathsf{f}(y,\mathsf{i}(\mathsf{f}(z_1,z_2))))) \to^+ \mathsf{f}(z_1,\mathsf{f}(z_2,\mathsf{f}(x,\mathsf{f}(y,\mathsf{f}(\mathsf{i}(z_2),\mathsf{i}(z_1)))))) =: u$. Now suppose we can ensure that the reduction order satisfies $\mathsf{f}(z_1,z_2) > \mathsf{i}(z_1)$ and $\mathsf{f}(z_1,z_2) > \mathsf{i}(z_1)$, and thus by transitivity $x, y > \mathsf{i}(z_1), \mathsf{i}(z_2)$. This admits the ordered rewrite steps

$$u \to^+ \mathsf{f}(z_1,\mathsf{f}(z_2,\mathsf{f}(\mathsf{i}(z_2),\mathsf{f}(\mathsf{i}(z_1),\mathsf{f}(x,y))))) \to \mathsf{f}(z_1,\mathsf{f}(\mathsf{i}(z_1),\mathsf{f}(x,y)))) \to \mathsf{f}(x,y)$$

So $(\star)$ is $\mathcal{F}$-ground-joinable if the constraints $\mathsf{f}(z_1,z_2) > \mathsf{i}(z_1), \mathsf{i}(z_2)$ are satisfied. Indeed the KBO with $\mathsf{i} > \mathsf{f} > \mathsf{0}$, $w_0 = w(\mathsf{0}) = 1$ and $w(\mathsf{i}) = w(\mathsf{f}) = 0$ orients $\mathcal{E}$ and satisfies these constraints. (For the signature $\mathcal{F}_\mathsf{a}$ some further case analysis is required, as indicated in Figure 1.)

Ground confluence of $\mathcal{E}^>$ is established by checking that the union of the joinability constraints that arise from equations in $\mathrm{CP}_>(\mathcal{E})$ can be satisfied by some reduction order that is compatible with $\mathcal{E}$. MædMax employs an SMT solver for this task and can indeed show ground confluence. The tools Waldmeister and E fail on this problem.

For a derivation tree as in Figure 1 it is intuitively clear that any grounding substitution corresponds to exactly one branch. This fact needed for the proof of Lemma 4 is made formal in Lemma 6. We write $G \Rightarrow_1 G'$ if there is some $\mathcal{G}$ such that $\{G\} \Rightarrow \mathcal{G}$ and $G' \in \mathcal{G}$.

Lemma 6. *Let $s\sigma, t\sigma$ be ground and $\gamma\colon \mathcal{G}_0 = \{(s \approx t, \varnothing, \varnothing)\} \Rightarrow^+ \mathcal{G}_n$. Then there is some $m \leqslant n$ such that for all $i < m$ there are $G_i \in \mathcal{G}_i$ such that $G_i = (s_i \approx t_i, \mathsf{O}_i, \tau_i)$ and (1) $s\tau_i = s_i$, $t\tau_i = t_i$ and there is some δ_i such that $\sigma = \tau_i\delta_i$, (2) $\succeq_{\mathsf{O}_i}$ covers δ_i, and (3) $G_i \Rightarrow_1 G_{i+1}$ for all $i < m$, and if $m < n$ then $\{G_m\} \Rightarrow \varnothing$. The sequence $G_0, \dots, G_m$ is called the* projection *of γ to σ.*

Proof of Lemma 4. Suppose σ is a substitution that is $\mathcal{F}$-grounding for s and t. By assumption $\mathcal{G}_0 \Rightarrow^+ \mathcal{G}_n$ holds for $\mathcal{G}_0 = \{(s \approx t, \varnothing, \varnothing)\}$ and $\mathcal{G}_n = \varnothing$. By Lemma 6 there is a projection $G_0, \dots, G_m$ to σ for some $m \leqslant n$, so for all $G_i = (s_i \approx t_i, O_i, \tau_i)$ with $i < m$ there is some δ_i satisfying $\sigma = \tau_i\delta_i$ and for all $x >_\mathsf{O} y$ in O_i it holds that $\delta_i(x) > \delta_i(y)$.

We show by induction on $\{s\sigma, t\sigma\}$ with respect to $>_\mathsf{mul}$ that $s\sigma \downarrow_{\mathcal{E}^>} t\sigma$ holds. In a nested induction on $m - i$ we establish $s_i\delta_i \downarrow_{\mathcal{E}^>} t_i\delta_i$. We first consider the base cases where $i = m$ and do a case distinction on the last step. If delete is applied then $s_i\delta_i \downarrow_{\mathcal{E}^>} s_i\delta_i$ trivially holds. In case of an equation step, if $s_i\delta = t_i\delta$ then joinability trivially holds. Otherwise $s_i \leftrightarrow_\mathcal{E} t_i$ implies $s_i\delta_i \to_{\mathcal{E}^>} t_i\delta_i$ or $t_i\delta_i \to_{\mathcal{E}^>} s_i\delta_i$ because of ground totality. Note that there cannot be an unsat step as $\succeq_{\mathsf{O}_i}$ covers σ. For the final base case, suppose subreducible was applied. As τ_i is reducible for some $x \in \mathcal{D}\mathrm{om}(\tau_i)$ also $\sigma = \tau_i\delta_i$ is reducible to, say, σ'. Then $s\sigma > s\sigma'$ or $t\sigma > t\sigma'$

must hold since ordered rewriting is compatible with $>$, so we have $\{s\sigma, t\sigma\} >_{\mathsf{mul}} \{s\sigma', t\sigma'\}$. Therefore the (outer) induction hypothesis yields $s\sigma' \downarrow_{\mathcal{E}^>} t\sigma'$, which implies $s\sigma \downarrow_{\mathcal{E}^>} t\sigma$.

In the step case where $i \neq m$ we can assume $s_{i+1}\delta_{i+1} \downarrow_{\mathcal{E}^>} t_{i+1}\delta_{i+1}$ by the (inner) induction hypothesis. First, suppose rewrite is applied, so $s_i \to_{\mathcal{E}^{\succ}\circ_i} s_{i+1}$ and $\delta_i = \delta_{i+1}$. By Lemma 2 this implies $s_i\delta_i \to_{\mathcal{E}^>} s_{i+1}\delta_i$. So $s_i\delta_i \downarrow_{\mathcal{E}^>} t_i\delta_i$ follows from $s_{i+1}\delta_{i+1} \downarrow_{\mathcal{E}^>} t_{i+1}\delta_{i+1}$. The second rewrite case is symmetric. Next, if orient is applied the statement is obvious unless $x\delta_i = y\delta_i$. But then $\rho\delta_{i+1} = \delta_i$, so $s_i\delta_i \downarrow_{\mathcal{E}^>} t_i\delta_i$ follows from $s_{i+1}\delta_{i+1} \downarrow_{\mathcal{E}^>} t_{i+1}\delta_{i+1}$. For instantiate the choice of G_i ensures $s = s_{i+1}\delta_{i+1} = s_i\delta_i$ and $t = t_{i+1}\delta_{i+1} = t_i\delta_i$.

We ultimately conclude that $s_0\sigma = s\sigma \downarrow_{\mathcal{E}^>} t\sigma = t_0\sigma$ holds. □

4 Conclusion

We presented a new criterion for ground joinability that can be exploited to check ground completeness. For the original signature, the resulting test is strictly more powerful than the approach by Martin and Nipkow.

The criterion is implemented in the tool MædMax[1], which implements an ordered-completion version of maximal completion [6]. A key characteristic of this approach is that a pool of equations is maintained, from which in every iteration a maximal terminating TRS $\mathcal{R}_i$ is extracted by exploiting maxSAT/maxSMT and a SAT/SMT-encoding of a reduction order (LPO, KBO, or a choice between the two). If $\mathcal{R}_i$ together with the remaining equations is ground confluent we are done, otherwise the new extended critical pairs are added and the procedure is reiterated.

As sketched in Example 5, the proposed criterion allows to express conditions on ground joinability as conditions on the reduction order. Moreover, LPO and KBO constraint solving is known to be NP-complete [9, 7], so satisfiability of order constraints can be SAT-encoded. This fact is exploited in MædMax to apply the unsat rule. Experiments show that our test allows MædMax to gain power over other tools, the results can be found on-line.

References

[1] J. Avenhaus, T. Hillenbrand, and B. Löchner. On using ground joinable equations in equational theorem proving. *JSC*, 36(1–2):217–233, 2003.

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

[3] L. Bachmair, N. Dershowitz, and D.A. Plaisted. Completion without failure. In H. Aït Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 2: Rewriting Techniques of *Progress in Theoretical Computer Science*, pages 1–30. Academic Press, 1989.

[4] H. Comon, P. Narendran, R. Nieuwenhuis, and M. Rusinowitch. Deciding the confluence of ordered term rewrite systems. *TOCL*, 4(1):33–55, 2003.

[5] D. Kapur, P. Narendran, and F. Otto. On ground-confluence of term rewriting systems. *Inform. Comput.*, 86(1):14–31, 1990.

[6] D. Klein and N. Hirokawa. Maximal completion. In *Proc. RTA-22*, volume 10 of *LIPIcs*, pages 71–80, 2011.

[7] K. Korovin and A. Voronkov. Knuth–Bendix constraint solving is NP-complete. *TOCL*, 6(2):361–388, 2005.

[8] U. Martin and T. Nipkow. Ordered Rewriting and Confluence. In *Proc. CADE-10*, volume 449 of *LNCS*, pages 366–380, 1990.

[9] R. Nieuwenhuis. Simple LPO constraint solving methods. *Inform. Process. Lett.*, 47(2):65–69, 1993.

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

Formalized Ground Completion*

Aart Middeldorp and Christian Sternagel

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

Abstract

Completion is the process of turning a set of equations into an equivalent confluent and terminating set of rewrite rules. It is known that completion will always succeed if the input equations are ground and the employed reduction order is total on (equivalent) ground terms. Moreover, every reduced ground rewrite system can be obtained by completion from any equivalent set of ground equations. We present the first formalized correctness proofs of these results.

1 Introduction

We assume familiarity with term rewriting [1] but recall the inference rules of abstract completion.

Definition 1. *The inference system KB of abstract (Knuth-Bendix) completion operates on pairs $\mathcal{E}, \mathcal{R}$ of equations $\mathcal{E}$ and rules $\mathcal{R}$. It consists of the following inference rules:*

deduce	$\dfrac{\mathcal{E}, \mathcal{R}}{\mathcal{E} \cup \{s \approx t\}, \mathcal{R}}$	*if* $s \mathrel{{}_{\mathcal{R}}\!\leftarrow \cdot \rightarrow_{\mathcal{R}}} t$	*compose*	$\dfrac{\mathcal{E}, \mathcal{R} \uplus \{s \to t\}}{\mathcal{E}, \mathcal{R} \cup \{s \to u\}}$	*if* $t \to_{\mathcal{R}} u$
orient	$\dfrac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{s \to t\}}$	*if* $s > t$	*simplify*	$\dfrac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E} \cup \{u \approx t\}, \mathcal{R}}$	*if* $s \to_{\mathcal{R}} u$
	$\dfrac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{t \to s\}}$	*if* $t > s$		$\dfrac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E} \cup \{s \approx u\}, \mathcal{R}}$	*if* $t \to_{\mathcal{R}} u$
delete	$\dfrac{\mathcal{E} \uplus \{s \approx s\}, \mathcal{R}}{\mathcal{E}, \mathcal{R}}$		*collapse*	$\dfrac{\mathcal{E}, \mathcal{R} \uplus \{t \to s\}}{\mathcal{E} \cup \{u \approx s\}, \mathcal{R}}$	*if* $t \to_{\mathcal{R}} u$

Here $>$ is an arbitrary but fixed reduction order. The inference system KB^- consists of the inference rules of KB except for deduce.

Snyder [8] proved that ground sets of equations (also called equational systems or ESs for short) can always be completed by KB^-. In the next section we present a proof of this result that we formalized in Isabelle/HOL [5]. Snyder further proved that every reduced ground rewrite system is canonical and can be obtained by completion from any equivalent set of ground equations, our formalization of which is the topic of Section 3.

Our formalization is part of IsaFoR [9][1] version 2.31 where it is located in the file `thys/Abstract_Completion/Ground_Completion.thy`. Furthermore all definitions, theorems, and lemmas in the PDF version of this manuscript are active hyperlinks to a (human readable) HTML presentation of our formalization.

We conclude this introduction with an example illustrating the inference system KB^- on a set of ground equations.

*This work is supported by the Austrian Science Fund (FWF): projects P27502 and P27528.

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

Example 1. *Consider the ES $\mathcal{E}$ consisting of the ground equations*

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a}))) \approx \mathsf{f}(\mathsf{b}) \qquad \mathsf{f}(\mathsf{f}(\mathsf{b})) \approx \mathsf{c} \qquad \mathsf{f}(\mathsf{c}) \approx \mathsf{a} \qquad \mathsf{f}(\mathsf{a}) \approx \mathsf{f}(\mathsf{f}(\mathsf{b}))$$

As reduction order we take LPO induced by the total precedence $\mathsf{a} > \mathsf{b} > \mathsf{c} > \mathsf{f}$. We start by applying orient *to the last two equations:*

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a}))) \approx \mathsf{f}(\mathsf{b}) \qquad \mathsf{f}(\mathsf{f}(\mathsf{b})) \approx \mathsf{c} \qquad \mathsf{f}(\mathsf{c}) \leftarrow \mathsf{a} \qquad \mathsf{f}(\mathsf{a}) \rightarrow \mathsf{f}(\mathsf{f}(\mathsf{b}))$$

An application of collapse *produces*

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a}))) \approx \mathsf{f}(\mathsf{b}) \qquad \mathsf{f}(\mathsf{f}(\mathsf{b})) \approx \mathsf{c} \qquad \mathsf{f}(\mathsf{c}) \leftarrow \mathsf{a} \qquad \mathsf{f}(\mathsf{f}(\mathsf{c})) \approx \mathsf{f}(\mathsf{f}(\mathsf{b}))$$

Next we orient the second equation:

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a}))) \approx \mathsf{f}(\mathsf{b}) \qquad \mathsf{f}(\mathsf{f}(\mathsf{b})) \rightarrow \mathsf{c} \qquad \mathsf{f}(\mathsf{c}) \leftarrow \mathsf{a} \qquad \mathsf{f}(\mathsf{f}(\mathsf{c})) \approx \mathsf{f}(\mathsf{f}(\mathsf{b}))$$

Two applications of simplify *produce*

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{c}))) \approx \mathsf{f}(\mathsf{b}) \qquad \mathsf{f}(\mathsf{f}(\mathsf{b})) \rightarrow \mathsf{c} \qquad \mathsf{f}(\mathsf{c}) \leftarrow \mathsf{a} \qquad \mathsf{f}(\mathsf{f}(\mathsf{c})) \approx \mathsf{c}$$

We continue by orienting the last equation:

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{c}))) \approx \mathsf{f}(\mathsf{b}) \qquad \mathsf{f}(\mathsf{f}(\mathsf{b})) \rightarrow \mathsf{c} \qquad \mathsf{f}(\mathsf{c}) \leftarrow \mathsf{a} \qquad \mathsf{f}(\mathsf{f}(\mathsf{c})) \rightarrow \mathsf{c}$$

Two applications of simplify *produce*

$$\mathsf{c} \approx \mathsf{f}(\mathsf{b}) \qquad \mathsf{f}(\mathsf{f}(\mathsf{b})) \rightarrow \mathsf{c} \qquad \mathsf{f}(\mathsf{c}) \leftarrow \mathsf{a} \qquad \mathsf{f}(\mathsf{f}(\mathsf{c})) \rightarrow \mathsf{c}$$

Orienting the remaining equation followed by a collapse step produces

$$\mathsf{c} \leftarrow \mathsf{f}(\mathsf{b}) \qquad \mathsf{f}(\mathsf{c}) \approx \mathsf{c} \qquad \mathsf{f}(\mathsf{c}) \leftarrow \mathsf{a} \qquad \mathsf{f}(\mathsf{f}(\mathsf{c})) \rightarrow \mathsf{c}$$

Finally, we orient the only remaining equation and collapse*,* compose*,* simplify*, and* delete *exhaustively, thereby obtaining the TRS $\mathcal{R}$*

$$\mathsf{c} \leftarrow \mathsf{f}(\mathsf{b}) \qquad \mathsf{f}(\mathsf{c}) \rightarrow \mathsf{c} \qquad \mathsf{c} \leftarrow \mathsf{a}$$

which constitutes a canonical presentation of $\mathcal{E}$.

2 Correctness

The absence of deduce from KB^- does not hurt for ground systems. If $s \leftarrow \cdot \rightarrow t$ and the two contracted redexes are at parallel positions then trivially $s \rightarrow \cdot \leftarrow t$. If the steps are identical then $s = t$. In the remaining case one of the contracted redexes is a subterm of the other contracted redex, and the effect of deduce is achieved by the collapse inference rule.

On the contrary, the absence of deduce is crucial to conclude that KB^- derivations are always finite.

Lemma 1. *There are no infinite sequences $\mathcal{E}_0, \varnothing \vdash_{\mathsf{KB}^-} \mathcal{E}_1, \mathcal{R}_1 \vdash_{\mathsf{KB}^-} \cdots$ for finite ground ESs $\mathcal{E}_0$.*

Proof. Let $\succ$ denote the lexicographic combination of the multiset extension $>_{\mathsf{mul}}$ of the reduction order $>$ with the standard order on natural numbers $>_{\mathbb{N}}$. Furthermore let $M(\mathcal{E},\mathcal{R})$ denote the (finite) multiset of left-hand sides and right-hand sides occurring in $\mathcal{E}$ and $\mathcal{R}$

$$M(\mathcal{E},\mathcal{R}) = \bigcup\{\{s,t\} \mid (s,t) \in \mathcal{E}\} \cup \bigcup\{\{s,t\} \mid (s,t) \in \mathcal{R}\}$$

and consider the function P that maps the pair $(\mathcal{E},\mathcal{R})$ to $(M(\mathcal{E},\mathcal{R}),|\mathcal{E}|)$. Now it is straightforward to verify that any infinite $\vdash_{\mathsf{KB}^-}$-sequence would give rise to an infinite sequence $P(\mathcal{E}_0,\varnothing) \succ P(\mathcal{E}_1,\mathcal{R}_1) \succ \cdots$, contradicting the well-foundedness of $\succ$. □

The formalization of the following preliminary result is covered by previous work [2].

Lemma 2. *If* $(\mathcal{E},\mathcal{R}) \vdash^*_{\mathsf{KB}} (\mathcal{E}',\mathcal{R}')$ *then* $\xleftrightarrow[\mathcal{E}\cup\mathcal{R}]{*} = \xleftrightarrow[\mathcal{E}'\cup\mathcal{R}']{*}$. □

Theorem 1. *If* $>$ *is total on* $\mathcal{E}$*-equivalent ground terms then every maximal* KB^- *run produces an equivalent canonical presentation for ground ES* $\mathcal{E}$*.*

Proof. Consider a maximal KB^- run $\mathcal{E}_0,\varnothing \vdash \mathcal{E}_1,\mathcal{R}_1 \vdash \cdots \vdash \mathcal{E}_n,\mathcal{R}_n$ where $\mathcal{E}_0 = \mathcal{E}$ is a ground ES. Because the run is maximal, no inference rule of KB^- is applicable to the final pair $(\mathcal{E}_n,\mathcal{R}_n)$. In particular, compose and collapse are not applicable and hence the final TRS $\mathcal{R}_n$ is reduced. Since $\mathcal{R}_n$ is also ground, it is canonical. From Lemma 2 and the inclusion $\mathsf{KB}^- \subseteq \mathsf{KB}$ we infer that $\mathcal{E}$ and $\mathcal{E}_n \cup \mathcal{R}_n$ are equivalent. It follows that $>$ is total on $\mathcal{E}_n$-equivalent ground terms and thus $\mathcal{E}_n = \varnothing$, for otherwise the run could be extended with an application of delete or simplify. Hence $\mathcal{R}_n$ and $\mathcal{E}$ are equivalent. □

The restriction on the reduction order $>$ in the above correctness theorem is easy to satisfy. In particular, it holds for any LPO or KBO based on a total precedence.

3 Completeness

The final result of this note states the completeness of ground completion. The proof makes use of the following preliminary results. The formalization of the first one is detailed in previous work [3].

Theorem 2 (Métivier [4])**.** *Let* $\mathcal{R}$ *and* $\mathcal{S}$ *be equivalent canonical TRSs. If* $\mathcal{R}$ *and* $\mathcal{S}$ *are compatible with the same reduction order then* $\mathcal{R} \doteq \mathcal{S}$*.* □

Here $\mathcal{R} \doteq \mathcal{S}$ denotes that $\mathcal{R}$ and $\mathcal{S}$ are identical modulo renaming of variables (that is, each rule of $\mathcal{R}$ has a variant in $\mathcal{S}$ and vice versa). The next concept is useful in the analysis of rewrite strategies [7]. It generalizes a number of earlier concepts, including the property $\leftarrow \cdot \rightarrow \subseteq \rightarrow \cdot \leftarrow \cup =$ which is known as WCR^1 and true for left-reduced ground TRSs.

Definition 2. *A TRS* $\mathcal{R}$ *has* random descent *if for every conversion* $a \leftrightarrow^* b$ *with normal form* b *we have* $a \rightarrow^n b$ *with* $n + l = r$*. Here* l *(*r*) denotes the number of* $\leftarrow$ *(*$\rightarrow$*) steps in the conversion* $a \leftrightarrow^* b$*.*

Theorem 3 (van Oostrom [6])**.** *Let* $\mathcal{R}$ *be a TRS with random descent. If* $a \leftrightarrow^* b$ *with normal form* b *then* a *is complete and all rewrite sequences from* a *to* b *have the same length.*

The short and direct proof given below has been formalized.

Proof. Let l (r) be the number of $\leftarrow$ ($\rightarrow$) steps in the conversion from a to b. We have $l \leqslant r$ since $n + l = r$ for some n by random descent. First we prove termination of a. For a proof by contradiction, suppose the existence of an infinite rewrite sequence

$$a = a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow \cdots$$

Clearly, $a \rightarrow^{r-l} a_{r-l}$ and thus there exists a conversion $a_{r-l} \,{}^*\!\leftarrow a \leftrightarrow^* b$ with r backwards and r forwards steps. Hence $a_{r-l} = b$ by another application of random descent and therefore $b \rightarrow a_{r-l+1}$, contradicting the fact that b is a normal form. Next we prove confluence of a. Suppose $c \,{}^*\!\leftarrow a \rightarrow^* d$. We obtain the two conversions $c \leftrightarrow^* b$ and $d \leftrightarrow^* b$, which are transformed into $c \downarrow d$ by two applications of random descent. Finally, assume there are two rewrite sequences $a \rightarrow^m b$ and $a \rightarrow^n b$ from a to b of length m and n. Reversing the first sequence and appending the second one yields a conversion $b \leftrightarrow^* b$ with m backwards and n forwards steps. A final application of random descent yields $b \rightarrow^k b$ for some k with $k + m = n$. Since b is a normal form, $k = 0$ and thus $m = n$ as desired. □

Lemma 3. *Reduced ground TRSs are canonical and have random descent.*

Proof. First we show that every left-reduced ground TRS $\mathcal{R}$ has random descent. To this end let $s \leftrightarrow^* t$ be a conversion between s and the normal form t. Now we proceed by induction on the length of the conversion. If it is empty or the first step is to the right, we are done. Otherwise, we have $s \leftarrow u \leftrightarrow^* t$ where the conversion has l (r) left-steps (right-steps) and obtain $u \rightarrow^k t$ with $k + l = r$ by the induction hypothesis. The remainder of the proof proceeds by induction on k together with the observation that left-reduced ground TRSs enjoy the WCR^1 property.

Moreover, every right-reduced ground TRS $\mathcal{R}$ is terminating. For assuming non-termination there would be a minimal non-terminating term t. This means that after a finite number of non-root steps $t \rightarrow^* u$ there will be a root-step $u \rightarrow v$ such that v is non-terminating. But since $\mathcal{R}$ is right-reduced and ground, v is a ground normal form.

Since all terms are terminating, confluence of $\mathcal{R}$ is an immediate consequence of the definition of random descent. □

Theorem 4. *For every ground ES $\mathcal{E}$ and every equivalent reduced ground TRS $\mathcal{R}$ there exist a reduction order $>$ and a derivation $\mathcal{E}, \varnothing \vdash_{\mathsf{KB}^-} \cdots \vdash_{\mathsf{KB}^-} \varnothing, \mathcal{R}$.*

Proof. Let $>$ be a reduction order that contains $\mathcal{R}$ and is total on $\mathcal{E}$-equivalent ground terms. Consider a maximal KB^- run starting from $\mathcal{E}$ and using $>$. According to Theorem 1, the run produces an equivalent reduced TRS $\mathcal{R}'$. Since $\mathcal{R} \subseteq >$ and $\mathcal{R}' \subseteq >$, we obtain $\mathcal{R} = \mathcal{R}'$ from Theorem 2. It remains to show that $>$ exists. Let $\sqsupset$ be a total precedence and define $s > t$ if and only if $s \leftrightarrow^*_{\mathcal{E}} t$ and either $d_{\mathcal{R}}(s) > d_{\mathcal{R}}(t)$ or both $d_{\mathcal{R}}(s) = d_{\mathcal{R}}(t)$ and $s \sqsupset_{\mathsf{lpo}} t$.[2] Here $d_{\mathcal{R}}(u)$ is the number of rewrite steps in $\mathcal{R}$ to normalize the term u, which is well-defined since all normalizing sequences in a reduced ground TRS have the same length as a consequence of Lemma 3 and Theorem 3. It is easy to show that $>$ has the required properties. The only interesting cases are closure under contexts and substitutions. Both are basically handled by the following observation: $d_{\mathcal{R}}(C[t\sigma]) = d_{\mathcal{R}}(C[t{\downarrow}\sigma]) + d_{\mathcal{R}}(t)$ for any term t (which holds due to random descent together with termination). This allows us to lift $d_{\mathcal{R}}(s) = d_{\mathcal{R}}(t)$ and $d_{\mathcal{R}}(s) > d_{\mathcal{R}}(t)$ into arbitrary contexts and substitutions. □

The above result cannot be generalized to left-linear right-ground systems, as shown in the following example due to Dominik Klein (personal communication).

[2] In the formalization we actually use $\sqsupset_{\mathsf{kbo}}$ with all weights set to 1, since in contrast to LPO, for KBO ground-totality for total precedences has already been formalized before.

Example 2. *Consider the ES* $\mathcal{E}$ *consisting of the two equations* $\mathsf{f}(x) \approx \mathsf{f}(\mathsf{a})$ *and* $\mathsf{f}(\mathsf{b}) \approx \mathsf{b}$*. Let* $>$ *be a reduction order. If* $\mathsf{f}(\mathsf{b}) > \mathsf{b}$ *does not hold, no inference rule of KB is applicable to* $(\mathcal{E}, \varnothing)$*. If* $\mathsf{f}(\mathsf{b}) > \mathsf{b}$ *then the second equation can be oriented*

$$(\mathcal{E}, \varnothing) \;\vdash\; (\{\mathsf{f}(x) \approx \mathsf{f}(\mathsf{a})\}, \{\mathsf{f}(\mathsf{b}) \to \mathsf{b}\})$$

At this point trivial equations of the shape $\mathsf{f}^n(\mathsf{b}) \approx \mathsf{f}^n(\mathsf{b})$ *with* $n \geqslant 0$ *can be deduced and subsequently deleted. No other possibilities exist and hence completion will fail on* $\mathcal{E}$*. Nevertheless, the TRS* $\mathcal{R}$ *consisting of the rewrite rule* $\mathsf{f}(x) \to \mathsf{b}$ *constitutes a canonical presentation of* $\mathcal{E}$*.*

References

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

[2] N. Hirokawa, A. Middeldorp, and C. Sternagel. A new and formalized proof of abstract completion. In *Proc. 5th International Conference on Interactive Theorem Proving*, volume 8558 of *Lecture Notes in Computer Science*, pages 292–307, 2014. doi: 10.1007/978-3-319-08970-6_19.

[3] N. Hirokawa, A. Middeldorp, C. Sternagel, and S. Winkler. Infinite runs in abstract completion. In *Proc. 2nd International Conference on Formal Structures for Computation and Deduction*, volume 84 of *Leibniz International Proceedings in Informatics*, pages 19:1–19:16, 2017. To appear.

[4] Y. Métivier. About the rewriting systems produced by the Knuth-Bendix completion algorithm. *Information Processing Letters*, 16(1):31–34, 1983. doi: 10.1016/0020-0190(83)90009-1.

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

[6] V. van Oostrom. Random descent. In *Proc. 18th International Conference on Rewriting Techniques and Applications*, volume 4533 of *Lecture Notes in Computer Science*, pages 314–328, 2007. doi: 10.1007/978-3-540-73449-9_24.

[7] V. van Oostrom and Y. Toyama. Normalisation by random descent. In *Proc. 1st International Conference on Formal Structures for Computation and Deduction*, volume 52 of *Leibniz International Proceedings in Informatics*, pages 32:1–32:18, 2016. doi: 10.4230/LIPIcs.FSCD.2016.32.

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

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

Z for Call-by-Value

Koji Nakazawa[1], Ken-etsu Fujita[2], and Yuta Imagawa[1]

[1] Graduate School of Informatics, Nagoya University, Nagoya, JAPAN
knak@i.nagoya-u.ac.jp, imagawa.yuuta@d.mbox.nagoya-u.ac.jp
[2] Graduate School of Science and Technology, Gunma University, Gunma, JAPAN
fujita@cs.gunma-u.ac.jp

Abstract

This work gives a new proof of the confluence of Carraro and Guerrieri's call-by-value lambda calculus λ_v^σ with permutation rules by the extended Z theorem, the compositional Z theorem.

1 Introduction

In general, it is hard to prove the confluence of higher-order rewriting, including (extensions of) the lambda calculus. There is a long history to give simple (or elegant) proofs for the confluence of higher-order rewriting systems and the lambda calculi: Church and Rosser's proof with the notion of the residuals of redexes, Tait and Martin-Löf's proofs with the parallel reduction, and Takahashi's proof with the complete development. Dehornoy and van Oostrom's Z theorem in [5] is one of the newest techniques which is widely applicable to confluence proofs in general setting. The Z theorem says that confluence of abstract rewriting system follows from existence of a mapping satisfying the Z property: any one-step reduction $a \to b$ implies $b \to^* f(a) \to^* f(b)$. This theorem has been applied to some variants of the lambda calculus in [5, 6, 2, 8].

The Z theorem is further generalized by Nakazawa and Fujita as the compositional Z theorem in [7], which enables us to use the Z theorem with dividing rewriting systems into two or more subsystems. The compositional Z gives a simple proof of confluence of the lambda calculus with permutation rules for direct sums. Ando's original confluence proof in [3] for that system is much complicated, and we cannot naïvely apply the original Z theorem because it seems hard to directly define a mapping satisfying the Z property for both beta and permutation reductions.

Carraro and Guerrieri's lambda calculus λ_v^σ in [4] is a call-by-value variant of the lambda calculus, in which they adopt permutation rules to avoid that reductions unexpectedly get stuck. In [1], this calculus is also called the shuffling calculus λ_{shuf}, and further discussed as one of call-by-value variants of the lambda calculus for open terms. As for the permutation rules for direct sums, the permutation rules of λ_v^σ make the confluence proof much harder, and we cannot straightforwardly adapt the ordinary proof techniques with the parallel reduction or the complete development. Carraro and Guerrieri proved the confluence of λ_v^σ by the commutativity of the β_v reduction and the permutation rules.

In this work, we show that the compositional Z theorem can be applied to λ_v^σ. As in [7], we cannot naïvely adapted the original Z theorem for λ_v^σ, and the problem can be avoided by the compositional Z. Although the outline of our proof follows [7], the main difference is that the mapping we consider here may leave redexes of permutation reductions, because the calculus has two kinds (directions) of permutation rules. Hence, we cannot apply the simplified variant of the compositional Z (Corollary 2.4 in [7]).

B. Accatoli and B. Felgenhauer (eds.); IWC 2017, pp. 57–61

2 λ_v^σ

In this section, the call-by-value lambda calculus λ_v^σ is introduced.

The values and terms of λ_v^σ are given as follows.

$$\begin{aligned} V &::= x \mid \lambda x.M && \text{(values)} \\ M &::= V \mid MM && \text{(terms)} \end{aligned}$$

The reduction rules of λ_v^σ are given as follows.

$$\begin{aligned} (\lambda x.M)V &\to_{\beta_v} [V/x]M && \\ (\lambda x.M)NL &\to_{\sigma_1} (\lambda x.ML)N && (x \in FV(L)) \\ V((\lambda x.M)N) &\to_{\sigma_3} (\lambda x.VM)N && (x \in FV(V)) \end{aligned}$$

Here, V denotes values, M, N, and L denote terms, and $[V/x]M$ is the usual capture-avoiding substitution. $FV(M)$ denotes the set of free variables in M. We consider the compatible closure of the reduction rules.

This calculus is introduced for operational characterization of solvability in call-by-value lambda calculi in particular for open terms. In Plotkin's original call-by-value lambda calculus λ_v in [9], the term $M \equiv (\lambda yx.xx)(zz)(\lambda x.xx)$ is stuck since zz is not a value, whereas semantically it is equivalent to $(\lambda x.xx)(\lambda x.xx)$ and hence unsolvable. In λ_v^σ, M is reduced to $(\lambda y.(\lambda x.xx)(\lambda x.xx))(zz)$ by the σ-rules, and we can discover the redex $(\lambda x.xx)(\lambda x.xx)$.

3 Compositional Z

We summarize Dehornoy and van Oostrom's Z theorem [5], and then extend it for compositional functions, called the *compositional Z* [7]. It gives a sufficient condition for that a compositional function satisfies the Z property, and enables us to consider a reduction system by dividing into two parts to prove confluence.

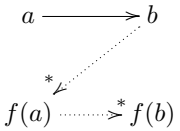
Definition 3.1 ((Weak) Z property)**.** Let $(A, \to)$ be an abstract rewriting system, and $\to^*$ be the reflexive transitive closure of $\to$. Let $\to_{\mathsf{x}}$ be another relation on A, and $\to_{\mathsf{x}}^*$ be its reflexive transitive closure.

1. A mapping f satisfies the *weak Z property for* $\to$ *by* $\to_{\mathsf{x}}$ if $a \to b$ implies $b \to_{\mathsf{x}}^* f(a) \to_{\mathsf{x}}^* f(b)$ for any $a, b \in A$.

2. A mapping f satisfies the *Z property for* $\to$ if it satisfies the weak Z property by $\to$ itself.

When f satisfies the (weak) Z property, we also say that f *is (weakly) Z.*

It becomes clear why we call it the Z property when we draw the condition as the following diagram.



Theorem 3.2 (Z theorem [5])**.** *If there exists a mapping satisfying the Z property for an abstract rewriting system, then it is confluent.*

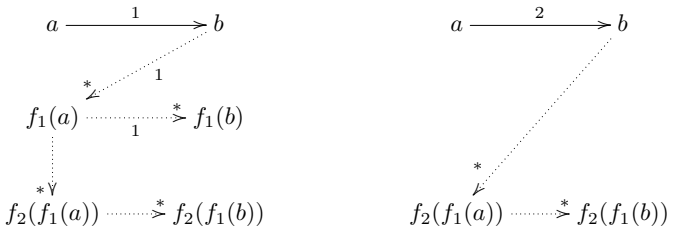


Figure 1: Proof of Theorem 3.3

In fact, we can often prove that the usual complete developments have the Z property.

The compositional Z is the following, which is easily proved from Theorem 3.2 with the diagrams in Figure 1.

Theorem 3.3 (Compositional Z [7]). *Let $(A, \to)$ be an abstract rewriting system, and $\to$ be $\to_1 \cup \to_2$. If there exist mappings $f_1, f_2 : A \to A$ such that*
(a) f_1 is Z for $\to_1$
(b) $a \to_1 b$ implies $f_2(a) \to^* f_2(b)$
(c) $a \to^* f_2(a)$ holds for any $a \in \mathrm{Im}(f_1)$
(d) $f_2 \circ f_1$ is weakly Z for $\to_2$ by $\to$,
then $f_2 \circ f_1$ is Z for $(A, \to)$, and hence $(A, \to)$ is confluent.

One of the simplest applications of the compositional Z is for the $\beta\eta$-reduction on the untyped lambda calculus (although it can be directly proved by the Z theorem as in [6]). In [7], the compositional Z is used for the lambda calculus with permutation rules for direct sums such as

$$(\mathsf{case}\ M\ \mathsf{with}\ \mathsf{inl}\ x_1 \Rightarrow N_1 \mid\ \mathsf{inr}\ x_2 \Rightarrow N_2)L \to \mathsf{case}\ M\ \mathsf{with}\ \mathsf{inl}\ x_1 \to N_1 L \mid\ \mathsf{inr}\ x_2 \to N_2 L,$$

which is denoted as $(M[x_1.N_1, x_2, N_2])L \to M[x_1.N_1L, x_2, N_2L]$ in [7]. Such permutation rules make confluence proofs much harder because of the critical pair: $M \equiv P[x_1.N_1, x_2.N_2][y_1.L_1, y_2.L_2]K$ is reduced to both

$$M_1 \equiv P[x_1.N_1[y_1.L_1, y_2.L_2], x_2.N_2[y_1.L_1, y_2.L_2]]K$$
$$M_2 \equiv P[x_1.N_1, x_2.N_2][y_1.L_1K, y_2.L_2K].$$

These can be reduced to a common term $M_3 \equiv P[x_1.N_1[y_1.L_1K, y_2.L_2K], x_2.N_2[y_1.L_1K, y_2.L_2K]]$, but in $M_1 \to^* M_3$ we have to reduce the redex which does not occur in M_1. Due to this fact, we can apply neither the ordinary parallel-reduction technique nor the original Z theorem.

4 Confluence of λ_v^σ by compositional Z

In this section, we see that λ_v^σ contains similar critical pairs to those in the lambda calculus with direct sums, and that the compositional Z solves the problem.

Let's consider the term $M \equiv (\lambda x.N)((\lambda y.K)L)P$, which is reduced to both

$$M_1 \equiv (\lambda x.NP)((\lambda y.K)L) \qquad (\text{by } \sigma_1)$$
$$M_2 \equiv (\lambda y.(\lambda x.N)K)LP \qquad (\text{by } \sigma_3).$$

These terms are reduced to a common term $M_3 \equiv (\lambda y.(\lambda x.NP)K)L$. $M_1 \to M_3$ is one-step σ_3 whereas $M_2 \to^* M_3$ consists of two σ_1 steps:

$$M_2 \equiv (\lambda y.(\lambda x.N)K)LP \to_{\sigma_1} (\lambda y.(\lambda x.N)KP)L \to_{\sigma_1} (\lambda y.(\lambda x.NP)K)L,$$

where the σ_1-redex of the second step does not occur in M_2. This means that we cannot naïvely extend the ordinary parallel reduction, by which M_2 is not reduced to M_3 in one step, and that, in the mapping satisfying the Z property, we have to reduce successive σ-reductions at once. For example, the above example shows that M must be mapped to M_3 or its reduct. We consider the auxiliary mapping $M@N$ to reduce successive σ-reductions such as

$$(\lambda y.(\lambda x.N)K)L@P \equiv (\lambda y.(\lambda x.N@P)K)L.$$

Here, the major difference from [7] is that there are two kinds of permutation rules σ_1 and σ_3 with opposite direction. We consider the following two auxiliary mappings $@_1$ and $@_3$:

$$\begin{array}{rlrl} (\lambda x.M)N@_1P &\equiv (\lambda x.M@_1P)N & V@_3(\lambda x.M)N &\equiv (\lambda x.V@_3M)N \\ M@_1P &\equiv MP \quad \text{(otherwise)} & V@_3M &\equiv VM \quad \text{(otherwise)} \end{array}$$

As for [7], the following naïve mapping does not satisfy the Z property.

$$\begin{aligned} x^* &\equiv x \\ (\lambda x.M)^* &\equiv \lambda x.M^* \\ ((\lambda x.M)V)^* &\equiv [V^*/x]M^* \\ (MN)^* &\equiv M^*@_1N^* \quad (M \text{ not a value}) \\ (VN)^* &\equiv V^*@_3N^* \quad \text{(otherwise)} \end{aligned}$$

For $P \equiv (\lambda x.xy)(\lambda z.z)v$ and $Q \equiv (\lambda x.xyv)(\lambda z.z)$, we have $P \to_{\sigma_1} Q$, but we also have $P^* \equiv (\lambda z.z)y@_1v \equiv (\lambda z.zv)y$ and $Q^* \equiv (\lambda z.z)yv$, and hence $P^* \not\to^* Q^*$.

In order to apply the compositional Z theorem, we divide the reductions of λ_v^σ into β_v and σ, and define the mapping $(\cdot)^{\mathsf{S}}$ and $(\cdot)^{\mathsf{B}}$ as follows.

$$\begin{array}{rlrl} x^{\mathsf{S}} &\equiv x & x^{\mathsf{B}} &\equiv x \\ (\lambda x.M)^{\mathsf{S}} &\equiv \lambda x.M^{\mathsf{S}} & (\lambda x.M)^{\mathsf{B}} &\equiv \lambda x.M^{\mathsf{B}} \\ (MN)^{\mathsf{S}} &\equiv M^{\mathsf{S}}@_1N^{\mathsf{S}} \quad (M \text{ not a value}) & ((\lambda x.M)V)^{\mathsf{B}} &\equiv [V^{\mathsf{B}}/x]M^{\mathsf{B}} \\ (VN)^{\mathsf{S}} &\equiv V^{\mathsf{S}}@_3N^{\mathsf{S}} & (MN)^{\mathsf{B}} &\equiv M^{\mathsf{B}}N^{\mathsf{B}} \quad \text{(otherwise)} \end{array}$$

Theorem 4.1. *The two mappings $(\cdot)^{\mathsf{S}}$ and $(\cdot)^{\mathsf{B}}$ (for σ- and β_v-reductions, respectively) satisfy the conditions for the compositional Z, and hence λ_v^σ is confluent.*

The outline of the proof almost follows [7]. However, in contrast to [7], the mapping $(\cdot)^{\mathsf{S}}$ does not necessarily collapse σ-steps, that is, there are terms M and N such that $M \to_\sigma N$ and $M^{\mathsf{S}} \to^+ N^{\mathsf{S}}$. For example, we have

$$M \equiv (\lambda x.x)(yz)((\lambda v.v)w) \to_{\sigma_1} (\lambda x.x((\lambda v.v)w))(yz) \equiv N,$$

and

$$\begin{aligned} M^{\mathsf{S}} &\equiv ((\lambda x.x)(yz))^{\mathsf{S}}@_1((\lambda v.v)w)^{\mathsf{S}} \equiv (\lambda x.x((\lambda v.v)w))(yz) \\ N^{\mathsf{S}} &\equiv (\lambda x.x((\lambda v.v)w))^{\mathsf{S}}@_3(yz)^{\mathsf{S}} \equiv (\lambda x.x@_3((\lambda v.v)w))(yz) \equiv (\lambda x.(\lambda v.xv)w)(yz). \end{aligned}$$

The σ_3-redex $x((\lambda v.v)w)$ in M^{S} is created in the application of $(\cdot)^{\mathsf{S}}$, and it is not reduced in M^{S}. Hence, we cannot apply the simplified variant of the compositional Z (Corollary 2.4 in [7]) for these mappings.

References

[1] Accattoli, B. and Guerrieri, G. Open call-by-value. In *Asian Symposium on Programming Languages and Systems (APLAS 2016)*, volume 10017 of *Lecture Notes in Computer Science*, pages 206–226, 2016.

[2] Accattoli, B. and Kesner, D. The permutative λ-calculus. In *Proceedings of the International Conference on Logic Programming and Automated Reasoning (LPAR 2012)*, volume 7180 of *Lecture Notes in Computer Science*, pages 15–22, 2012.

[3] Ando, Y. Church-Rosser property of a simple reduction for full first-order classical natural deduction. *Annals of Pure and Applied Logic*, 119:225–237, 2003.

[4] Carraro, A. and Guerrieri, G. A semantical and operational account of call-by-value solvability. In *Foundations of Software Science and Computation Structures (FoSSaCS 2014)*, volume 8412 of *Lecture Notes in Computer Science*, pages 103–18. Springer, 2014.

[5] Dehornoy, P. and van Oostrom, V. Z, proving confluence by monotonic single-step upperbound functions. In *Logical Models of Reasoning and Computation (LMRC-08)*, 2008.

[6] Komori, Y., Matsuda, N., and Yamakawa, F. A simplified proof of the church-rosser theorem. *Studia Logica*, 102(1):175–183, 2013.

[7] Nakazawa, K. and Fujita, K. Compositional Z: Confluence proofs for permutative conversion. *Studia Logica*, 104:1205–1224, 2016.

[8] Nakazawa, K. and Nagai, T. Reduction system for extensional lambda-mu calculus. In *25th International Conference on Rewriting Techniques and Applications joint with the 12th International Conference on Typed Lambda Calculi and Applications (RTA-TLCA 2014)*, volume 8560 of *Lecture Notes in Computer Science*, pages 349–363, 2014.

[9] Plotkin, G. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

Aspects of Layer Systems in IsaFoR*

Bertram Felgenhauer and Franziska Rapp

Department of Computer Science, University of Innsbruck, Austria
{bertram.felgenhauer|franziska.rapp}@uibk.ac.at

Abstract

We report on an ongoing formalization of layer systems in Isabelle.

1 Introduction

Toyama's theorem [6, 9] states that confluence is modular, i.e., that the union of two confluent term rewrite systems (TRSs) over disjoint signatures is confluent if and only if the two TRSs themselves are confluent. This opens up a decomposition approach to proving confluence, which is attractive, because different confluence criteria may apply to the constituent TRSs that do not apply to their union. By adapting the modularity proof, several other results have been proved. For example, confluence is preserved by currying [5]. Layer systems [3] were introduced as an abstraction from these proofs, which work by decomposing terms into a maximal top and remaining aliens. A layer system $\mathfrak{L}$ is simply the set of allowed tops; for modularity, those are homogeneous multi-hole contexts, i.e., multi-hole contexts whose function symbols all belong to the signature of only one of the two given TRSs. At the heart of layer systems lies yet another adaptation of the modularity proof in [6]. The main results correspond to the *if* direction of modularity as stated above. When establishing confluence by layer systems, as remaining proof obligations, one has to check that a layer system satisfies so called layer conditions, which is easier than doing a full adaptation of the modularity proof.

This note describes an ongoing effort to formalize layer systems, which, once complete, will enable certification of confluence proofs based on persistence and currying. In fact, the prospect of formalization was one of the selling points of layer systems; whereas adapting existing proofs is convenient on paper, it becomes a burden when done in a formalization; as with any code duplication in software engineering, it would increase maintenance costs and should therefore be avoided. We use Isabelle [7] for our formalization[1], building on top of IsaFoR [8].

Notation. We use notation from term rewriting. Let $\mathcal{F}$, $\mathcal{V}$ be a signature. Then $\mathcal{T}(\mathcal{F},\mathcal{V})$ is the set of terms over that signature; $\mathcal{C}(\mathcal{F},\mathcal{V})$ is the set of multihole contexts (which may contain occurrences of an extra constant $\Box$, denoting holes.) On multihole contexts, we have a partial order $\sqsubseteq$ which is generated by $\Box \sqsubseteq C$ and closure under contexts. The corresponding partial supremum operation is denoted by $\sqcup$; intuitively it merges multi-hole contexts.

2 Layer Conditions

Let $\mathfrak{L}$ be a set of multi-hole contexts, which we intend to use for decomposing terms. We recall the definitions of a layer system, and (weakly) layered TRSs.

Definition 1 ([3, Definition 3.1])**.** Let $\mathfrak{L} \subseteq \mathcal{C}(\mathcal{F},\mathcal{V})$ be a set of multi-hole contexts over $\mathcal{F}$. Then $L \in \mathfrak{L}$ is called a *top* of a context $C \in \mathcal{C}(\mathcal{F},\mathcal{V})$ (according to $\mathfrak{L}$) if $L \sqsubseteq C$. A top is a *max-top* of C if it is maximal with respect to $\sqsubseteq$ among the tops of C.

*This work is supported by FWF (Austrian Science Fund) project P27528.

[1] The theories can be viewed at http://cl-informatik.uibk.ac.at/software/lisa/iwc2017/

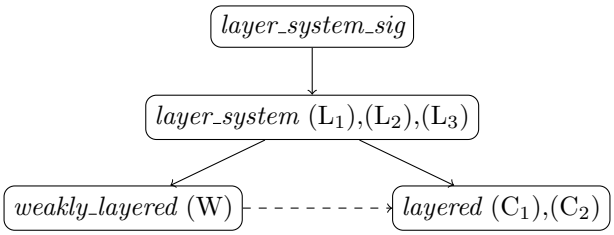


Figure 1: Hierarchy of locales.

Definition 2 ([3, Definition 3.3])**.** Let $\mathcal{F}$ be a signature. A set $\mathfrak{L} \subseteq \mathcal{C}(\mathcal{F}, \mathcal{V})$ of contexts is called a *layer system* if it satisfies properties (L_1), (L_2), and (L_3). The elements of $\mathfrak{L}$ are called *layers*. A TRS $\mathcal{R}$ over $\mathcal{F}$ is *weakly layered (according to a layer system $\mathfrak{L}$)* if condition (W) is satisfied for each $\ell \to r \in \mathcal{R}$. It is *layered (according to a layer system $\mathfrak{L}$)* if conditions (W), (C_1), and (C_2) are satisfied. The conditions are as follows:

(L_1) Each term in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ has a non-empty top.

(L_2) If $x \in \mathcal{V}$ and $C \in \mathcal{C}(\mathcal{F}, \mathcal{V})$ then $C[x]_p \in \mathfrak{L}$ if and only if $C[\Box]_p \in \mathfrak{L}$.

(L_3) If $L, N \in \mathfrak{L}$, $p \in \mathcal{P}os_{\mathcal{F}}(L)$, and $L|_p \sqcup N$ is defined then $L[L|_p \sqcup N]_p \in \mathfrak{L}$.

(W) If M is a max-top of s, $p \in \mathcal{P}os_{\mathcal{F}}(M)$, and $s \to_{p,\ell\to r} t$ then $M \to_{p,\ell\to r} L$ for some $L \in \mathfrak{L}$.

(C_1) In (W) either L is a max-top of t or $L = \Box$.

(C_2) If $L, N \in \mathfrak{L}$ and $L \sqsubseteq N$ then $L[N|_p]_p \in \mathfrak{L}$ for any $p \in \mathcal{P}os_{\Box}(L)$.

In Isabelle, we bundle these assumptions in locales [1]. For example, the first three layer conditions are formalized as follows:

```
locale layer_system_sig = fixes F :: 'f sig and L :: ('f, 'v) mctxt set

locale layer_system = layer_system_sig F L for F :: 'f sig and
  L :: ('f, 'v :: infinite) mctxt set +
  assumes L_sig: L ⊆ C
  and L1: t ∈ T ⟹ ∃ L ∈ L. L ≠ MHole ∧ L ≤ mctxt_of_term t
  and L2: p ∈ poss_mctxt C ⟹
    mreplace_at C p (MVar x) ∈ L ⟷ mreplace_at C p MHole ∈ L
  and L3: L ∈ L ⟹ N ∈ L ⟹ p ∈ funposs_mctxt L ⟹
    (subm_at L p, N) ∈ comp_mctxt ⟹ mreplace_at L p (subm_at L p ⊔ N) ∈ L
```

The first locale, *layer_system_sig*, is used to define $\mathcal{T}$ and $\mathcal{C}$, the set of terms and multi-hole contexts over $\mathcal{F}$, and the concept of max-tops. Actually max-tops are defined separately for terms and for multi-hole contexts, because while on paper, multi-hole contexts are just terms, in IsaFoR they have their own type. In total, four locales are defined, capturing the layer conditions, cf. Figure 1. Note that condition (W) is not part of the *layered* locale; it would be redundant because (C_1) implies (W). In Isabelle we have encoded this fact by proving that *layered* is a sublocale of *weakly_layered*, as indicated by the dashed arrow.

Using the layer system to decompose terms from the top yields the following notion of rank.

Definition 3 ([3, Definition 3.6])**.** Let $t = M[t_1, \ldots, t_n]$ with M the max-top of t. We define $\mathrm{rank}(t) = 1 + \max\{\mathrm{rank}(t_i) \mid 1 \leqslant i \leqslant n\}$, where $\max(\varnothing) = 0$ ($t_1, \ldots, t_n$ are the *aliens* of t).

The main theorems of [3] can be stated as follows (we omit [3, Theorem 4.3] to save space).

Theorem 4 ([3, Theorem 4.1])**.** *Let $\mathcal{R}$ be a weakly layered TRS that is confluent on terms of rank one. If $\mathcal{R}$ is left-linear then $\mathcal{R}$ is confluent.*

Theorem 5 ([3, Theorem 4.6])**.** *Let $\mathcal{R}$ be a layered TRS that is confluent on terms of rank one. Then $\mathcal{R}$ is confluent.*

Within the formalization, Theorem 4 will be established inside the *weakly_layered* locale, whereas Theorem 5 is expected to hold in the *layered* locale. The proofs of these main results correspond to Section 4 of [3], which we have fully formalized up to Lemma 4.18. The section goes up to Lemma 4.36, so a big chunk remains to be done. Nevertheless, we could already work on the applications like modularity and currying, because they are merely instantiations of these locales, which can be established independently of the main results.

3 Currying

Here we consider currying as one application of the layer framework, which we formalized in Isabelle. Instead of applying functions to several arguments at once, currying introduces a binary function symbol that is used for applying arguments to functions one by one. In functional programming, currying turns a function of type $A_1 \times \cdots \times A_n \to B$ into one of type $A_1 \to \cdots \to A_n \to B$, enabling partial application. For term rewriting systems (TRSs) we introduce a fresh function symbol $\bullet$ to denote application, whereas every other function symbol becomes a constant. By convention we write f_n to denote a function symbol of arity n. Moreover, we denote the arity of a function symbol f with respect to the signature $\mathcal{F}$ by $\mathsf{a}_{\mathcal{F}}(f)$. We identify $f_{\mathsf{a}_{\mathcal{F}}(f)}$ with f.

Definition 6. Given a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, its *curried version* $\mathsf{Cu}(\mathcal{R})$ consists of rules $\{\mathsf{Cu}(l) \to \mathsf{Cu}(r) \mid \ell \to r \in \mathcal{R}\}$, where $\mathsf{Cu}(t) = t$ if t is a variable and $\mathsf{Cu}(f(t_1, \ldots, t_n)) = f_0 \bullet \mathsf{Cu}(t_1) \bullet \cdots \bullet \mathsf{Cu}(t_n)$. Here $\bullet$ is a fresh left-associative function symbol.

Currying is useful for deciding properties such as confluence [2] or termination [4]. For analyzing confluence by currying, the following result is important.

Theorem 7. *Let $\mathcal{R}$ be a TRS. If $\mathcal{R}$ is confluent, then $\mathsf{Cu}(\mathcal{R})$ is confluent.*

This result was proved by Kahrs [5]. Rather than working directly with $\mathsf{Cu}(\mathcal{R})$, Kahrs works with the *partial parametrization* of $\mathcal{R}$, which is given by $\mathsf{PP}(\mathcal{R}) = \mathcal{R} \cup \mathcal{U}_{\mathcal{F}}$, where $\mathcal{U}_{\mathcal{F}}$ is the set of uncurrying rules for $\mathcal{F}$ (see Definition 8). Confluence of $\mathsf{PP}(\mathcal{R})$ and $\mathsf{Cu}(\mathcal{R})$ are closely related, cf. Lemma 9.

Definition 8. Given a signature $\mathcal{F}$, the *uncurrying* rules $\mathcal{U}_{\mathcal{F}}$ are rules $f_i(x_1, \ldots, x_i) \bullet x_{i+1} \to f_{i+1}(x_1, \ldots, x_{i+1})$ for every function symbol $f \in \mathcal{F}$ and $1 \leqslant i < \mathsf{a}_{\mathcal{F}}(f)$.

Lemma 9 ([5, Proposition 3.1])**.** *Let $\mathcal{R}$ be a TRS. Then $\mathsf{Cu}(\mathcal{R})$ is confluent if $\mathsf{PP}(\mathcal{R})$ is.*

Hence in order to prove Theorem 7 it suffices to prove that $\mathsf{PP}(\mathcal{R})$ is confluent. To this end, we make use of Theorem 5. Hence we need to show that $\mathsf{PP}(\mathcal{R})$ is layered according to some set of layers $\mathfrak{L}$ and confluent on terms of rank one. First of all we have to define a suitable set

of layers. We choose $\mathfrak{L} = \mathfrak{L}_1 \cup \mathfrak{L}_2$ where $\mathfrak{L}_1$ is the smallest extension of $\mathcal{V}_\Box = \mathcal{V} \cup \{\Box\}$ such that $f_m(s_1, \ldots, s_m) \bullet s_{m+1} \bullet \cdots \bullet s_n \in \mathfrak{L}_1$ for all $f \in \mathcal{F}$, $1 \leqslant m \leqslant n \leqslant \mathsf{a}_\mathcal{F}(f)$ and $s_1, \ldots, s_n \in \mathfrak{L}_1$, and $\mathfrak{L}_2 = \{x \bullet t \mid x \in \mathcal{V}_\Box \text{ and } t \in \mathfrak{L}_1\}$. This definition realizes a separation between well-formed terms ($\mathfrak{L}_1$), whose $\mathcal{U}_\mathcal{F}$-normal form contains no $\bullet$ symbol, and ill-formed terms ($\mathfrak{L}_2$), whose $\mathcal{U}_\mathcal{F}$-normal form contains exactly one $\bullet$ symbol at the root. As required for condition (L_1), variables and holes are treated interchangeably.

Whereas for Lemma 9 we could follow the lines of the paper proof, the formalization of the fact that $\mathsf{PP}(\mathcal{R})$ is layered according to $\mathfrak{L}$ turned out to be much more tedious. First of all, we found it convenient to define functions that *compute* the max-top of a term, since the abstract definition of max-tops in the layer framework is not really suitable for proofs in Isabelle.

Definition 10. The following function checks whether the number of arguments applied to the first non-$\bullet$ function symbol f is at most the arity $\mathsf{a}_\mathcal{F}(f)$ according to the original signature $\mathcal{F}$

$$\mathsf{check}(t, m) = \begin{cases} \mathit{false} & \text{if } t \in \mathcal{V} \\ \mathsf{check}(t_1, m+1) & \text{if } t = t_1 \bullet t_2 \\ \mathsf{a}_\mathcal{F}(f) \geqslant m + n & \text{if } t = f_n(t_1, \ldots, t_n) \end{cases}$$

Let $\mathcal{F}^\bullet = \mathcal{F} \cup \{\bullet\}$. The max-top mt_Cu of a term $t \in \mathcal{T}(\mathcal{F}^\bullet, \mathcal{V})$ with respect to $\mathfrak{L}$ is defined as

$$\mathsf{mt}_\mathsf{Cu}(t) = \begin{cases} \mathrm{t} & \text{if } t \in \mathcal{V} \\ f(\mathsf{mt}_1(t_1, 0), \ldots, \mathsf{mt}_1(t_n, 0)) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } (\mathsf{check}(t, 0) \text{ or } t_1 \in \mathcal{V}) \\ \Box \bullet \mathsf{mt}_1(t_2, 0) & \text{otherwise (in which case } t = t_1 \bullet t_2) \end{cases}$$

Here $\mathsf{mt}_1(t, m)$ computes the max-top of t with respect to $\mathfrak{L}_1$, where m is the number of already applied arguments:

$$\mathsf{mt}_1(t, m) = \begin{cases} \mathrm{t} & \text{if } t \in \mathcal{V} \\ \mathsf{mt}_1(t_1, m+1) \bullet \mathsf{mt}_1(t_2, 0) & \text{if } t = t_1 \bullet t_2 \text{ and } \mathsf{check}(t, m) \\ f(\mathsf{mt}_1(t_1, 0), \ldots, \mathsf{mt}_1(t_n, 0)) & \text{if } t = f(t_1, \ldots, t_n), f \neq \bullet \text{ and } \mathsf{check}(t, m) \\ \Box & \text{otherwise} \end{cases}$$

Note that there is some redundancy, since the check function does the same counting several times. However the definition is easier like this.

After proving the correctness of mt_1 and mt_Cu, the main difficulty was the proof of condition (C_1) for $\mathfrak{L}$ and $\mathsf{PP}(\mathcal{R})$. We sketch a constructive proof here, since this gives the best intuition and is also easiest to formalize in our opinion. In order to establish (C_1), we need to analyze a rewrite step $s = C[l\sigma]_p \to C[r\sigma]_p$ with p a function position of the max-top M of s; our goal is to obtain a rewrite step $M = D[l\sigma']_p \to D[r\sigma']_p$. The following two lemmas allow pushing the computation of the max-top (using mt_1) all the way to the substitution σ.

Lemma 11. *Let s be a term and p the hole position of context C such that $C[s]_p \in \mathcal{T}(\mathcal{F}^\bullet, \mathcal{V})$ and $p \in \mathcal{P}os_{\mathcal{F}^\bullet}(\mathsf{mt}_1(C[s], j))$. Then there exists a context D and a natural number k such that $\mathsf{mt}_1(C[s], j) = D[\mathsf{mt}_1(s, k)]$, and $\mathsf{mt}_1(C[t], j) = D[\mathsf{mt}_1(t, k)]$ for any term $t \in \mathcal{T}(\mathcal{F}^\bullet, \mathcal{V})$ having the same number of missing arguments as s.*

Lemma 12. *Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. Then $\mathsf{mt}_1(t \cdot \sigma, 0) = \mathsf{mt}_1(t, 0) \cdot \sigma'$ with $\sigma' = (\lambda x.\, \mathsf{mt}_1(x, 0)) \circ \sigma$.*

Using these two lemmas, we can obtain the desired rewrite step from M by the following computation, where for simplicity we only consider the case $M \in \mathfrak{L}_1$ and $l \to r \in \mathcal{R}$:

$$M = \mathsf{mt}_{\mathsf{Cu}}(s) = \mathsf{mt}_1(C[l \cdot \sigma], 0) \overset{11}{=} D[\mathsf{mt}_1(l \cdot \sigma, k)] \overset{12}{=} D[\mathsf{mt}_1(l, 0) \cdot \sigma'] = D[l \cdot \sigma']$$
$$\to_{p,\ell\to r} D[r \cdot \sigma'] = D[\mathsf{mt}_1(r, 0) \cdot \sigma'] \overset{12}{=} D[\mathsf{mt}_1(r \cdot \sigma, k)] \overset{11}{=} \mathsf{mt}_1(C[r \cdot \sigma], 0)$$

The uses of the previous two lemmas are indicated above the equalities. Note that the number of missing arguments of r is the same as for l, so we can use Lemma 11 in both directions. Furthermore $k = 0$, because $\mathsf{mt}_1(l \cdot \sigma, k) = \Box$ otherwise, so the rewrite step would not take place at a function position of M. Hence Lemma 12 is applicable. Furthermore, we use $\mathsf{mt}_1(l, 0) = l$ ($\mathsf{mt}_1(r, 0) = r$), using that l (r) is well-formed. If $C = \Box$, r is a variable and $\mathsf{check}(r \cdot \sigma)$ is false, $\mathsf{mt}_1(C[r \cdot \sigma], 0) = \Box$. Otherwise, the max-top of $C[r \cdot \sigma]$ is equal to $\mathsf{mt}_1(C[r \cdot \sigma], 0)$.

4 Conclusion

We have presented some aspects of a formalization of layer systems in Isabelle. Let us conclude with some statistics. In [3], the setup of the layer systems and the proof of the main results up to Lemma 4.18 spans approximately 150 lines of text. The corresponding formalization is about 3000 lines in length, corresponding to a *de Bruijn factor* of about 20. The section on currying in the original paper [3, Section 5.4], containing the proof of Theorem 7, covers about 80 lines (including 2 intermediate results in Kahrs [5]). Whereas only 9 definitions were needed to prepare for the proof, in the Isabelle formalization 24 definitions (abbreviations counted half) and 122 lemmas were necessary. Overall the formalization spans about 3200 lines of Isar code, which implies a de Bruijn factor of approximately 40. This high factor may be due to the fact that many case distinctions on the shape of terms were necessary and counting the number of applied arguments was tedious. Moreover, the formalization distinguishes terms from multi-hole contexts and hence several conversions between those types were necessary. Since the ultimate goal of our formalization effort is the certification of confluence proofs exploiting currying, we plan to finish the formalization of the layer framework and formalize further applications, most notably persistence of many-sorted TRSs.

References

[1] C. Ballarin. Locales: A module system for mathematical theories. *JAR*, 52(2):123–153, 2014.

[2] B. Felgenhauer. Deciding confluence of ground term rewrite systems in cubic time. In *Proc. 23rd RTA*, volume 15 of *LIPIcs*, pages 165–175, 2012.

[3] B. Felgenhauer, A. Middeldorp, H. Zankl, and V. van Oostrom. Layer systems for proving confluence. *ACM TOCL*, 16(2:14):1–32, 2015.

[4] N. Hirokawa, A. Middeldorp, and H. Zankl. Uncurrying for termination. In *Proc. 15th LPAR*, pages 667–681, 2008.

[5] S. Kahrs. Confluence of curried term-rewriting systems. *JSC*, 19(6):601–623, 1995.

[6] J.W. Klop, A. Middeldorp, Y. Toyama, and R. de Vrijer. Modularity of confluence: A simplified proof. *IPL*, 49:101–109, 1994.

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

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

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

ACP: System Description for CoCo 2017

Takahito Aoto[1] and Yoshihito Toyama[2]

[1] Faculty of Engineering, Niigata University
aoto@ie.niigata-u.ac.jp
[2] 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. ACP integrates multiple direct criteria for guaranteeing confluence of TRSs. It incorporates divide–and–conquer criteria by which confluence or non-confluence of TRSs can be inferred from those of their components. Several methods for disproving confluence are also employed. A list of implemented criteria and methods can be found on the website of ACP [1]. 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) and is provided as a heap image that can be loaded into SML/NJ runtime systems. It uses a SAT prover such as MiniSAT and an SMT prover YICES as external provers. It internally contains an automated (relative) termination prover for TRSs but external (relative) termination provers can be substituted optionally. The input TRS is specified in the (old) TPDB format. Users can specify criteria to be used so that each criterion or any combination of them can be tested. Several levels of verbosity are available for the output so that users can investigate details of the employed approximations for each criterion or can get only the final result of prover's attempt. For some criteria, it supports generation of proofs in CPF format that can be certified by certifiers. The source code and a list of implemented criteria are found on the webpage [1].

The internal structure of the prover is kept simple and is mostly inherited from the version 0.11a, which has been described in [2]. No new (non-)confluence criterion has been incorporated from the one submitted for CoCo 2016.

References

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

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

B. Accatoli and B. Felgenhauer (eds.); IWC 2017, pp. 69–69

ACPH: System Description for CoCo 2017

Kouta Onozawa[1], Kentaro Kikuchi[1], Takahito Aoto[2], and Yoshihito Toyama[1]

[1] RIEC, Tohoku University
{ onozawa, kentaro, toyama }@nue.riec.tohoku.ac.jp
[2] Faculty of Engineering, Niigata University
aoto@ie.niigata-u.ac.jp

Higher-order rewriting systems (HRSs) is a formalism of rewriting with variable binding and higher-order functions [2]. Higher-order rewriting deals with simply-typed lambda-terms with constants, which are identified modulo $\beta\eta$-equality. HRSs are a set of rewrite rules whose left-hand sides are restricted to patterns.

ACPH (Automated Confluence Prover for HRSs) is a tool for proving confluence of HRSs. If the tool succeeds to prove that an input HRS is confluent, it outputs YES. If the tool succeeds to prove that an input HRS is not confluent, it outputs NO. If the tool can not determine whether an input HRS is confluent or not, it outputs MAYBE. The tool uses following criteria for proving confluence and non-confluence of HRSs [1].

- If a HRS $\mathcal{R}$ is weakly orthogonal (left-linear and all critical pairs are trivial), then $\mathcal{R}$ is confluent.
- If a HRS $\mathcal{R}$ is terminating, then all critical pairs are joinable iff $\mathcal{R}$ is confluent.

The algorithms used in the program are based on those described in [1, 2]. For proving termination of HRSs, a higher-order termination tool WANDA[3] is used. ACPH program is written in Standard ML of New Jersey, and ACPH is provided as a heap image that can be loaded into SML/NJ runtime systems. It can be used from the command line by typing the following command:

```
$ sml @SMLload=acph.x86-linux <filename>
```

No new (non-)confluence criterion has been incorporated from the one submitted for CoCo 2016.

References

[1] Tobias Nipkow, Functional unification of higher-order patterns, *Proceedings of eighth annual IEEE symposium on logic in computer science*, pp.64-74, 1993.

[2] Richard Mayr, Tobias Nipkow, Higher-order rewrite systems and their confluence, *Theoretical computer science 192*, pp. 3-29, 1998.

[3] WANDA: A Higher-Order Termination Tool, http://wandahot.sourceforge.net/index.html

AGCP: System Description for CoCo 2017

Takahito Aoto[1] and Yoshihito Toyama[2]

[1] Faculty of Engineering, Niigata University
aoto@ie.niigata-u.ac.jp
[2] RIEC, Tohoku University
toyama@nue.riec.tohoku.ac.jp

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

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

References

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

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

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

CoCo 2017 Participant: CeTA 2.31*

Julian Nagele, Christian Sternagel, and Thomas Sternagel

Department of Computer Science, University of Innsbruck, Austria

Automatic provers have become popular in many areas like theorem proving, SMT, etc. Since such provers are complex pieces of software, they might contain errors that lead to wrong answers, i.e., incorrect proofs. Therefore, certification of the generated proofs is of major importance.

The tool CeTA [9] is a certifier that can be used to certify confluence and non-confluence proofs of term rewrite systems (TRSs) and conditional term rewrite systems (CTRSs). Its soundness is proven as part of IsaFoR, the *Isabelle Formalization of Rewriting*. The following techniques are currently supported in CeTA—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/ceta/).

Term rewrite systems. For terminating systems CeTA can check confluence via the critical pair lemma. For possibly non-terminating TRSs CeTA supports several criteria based on linearity and restricted joinability of critical pairs [3], the rule labeling heuristic [4], addition and removal of redundant rules [2], and, since 2017, terminating critical-pair-closing systems [5]. To certify non-confluence one can provide a divergence and a certificate for non-joinability, based on distinct normal forms, *tcap*, interpretations, or tree automata [1].

Conditional term rewrite systems. For CTRSs CeTA supports: certifying confluence of almost orthogonal, properly oriented, right-stable 3-CTRSs [6]; unraveling, a technique for transforming a given CTRS into a TRS; and, since 2017, certifying confluence of quasi-decreasing strongly deterministic CTRSs, possibly employing the new *inlining* technique [7]. Also new in 2017 is support for certifying non-confluence methods [8].

References

[1] B. Felgenhauer and R. Thiemann. Reachability, confluence, and termination analysis with state-compatible automata. *I&C* 253(3), 467–483, 2016.

[2] J. Nagele, B. Felgenhauer, and A. Middeldorp. Improving automatic confluence analysis of rewrite systems by redundant rules. In *RTA*, volume 36 of *LIPIcs*, pp. 257–268, 2015.

[3] J. Nagele and A. Middeldorp. Certification of classical confluence results for left-linear term rewrite systems. In *ITP*, volume 9807 of *LNCS*, pp. 290–306, 2016.

[4] J. Nagele, B. Felgenhauer, and H. Zankl. Certifying confluence proofs via relative termination and rule labeling. *LMCS* 13(2:4), 1–27, 2017.

[5] M. Oyamaguchi, and N. Hirokawa Confluence and critical-pair-closing systems. In Proc. 3rd *IWC*, pp. 29–33, 2014.

[6] C. Sternagel and T. Sternagel. Certifying confluence of almost orthogonal CTRSs via exact tree automata completion. In *FSCD*, volume 52 of *LIPIcs*, pp. 29:1–29:16, 2016.

[7] C. Sternagel and T. Sternagel. Certifying Confluence of Quasi-Decreasing Strongly Deterministic Conditional Term Rewrite Systems. In *CADE*, volume 10395 of *LNCS*, pp. 413–431, 2017.

[8] T. Sternagel and C. Sternagel. Certified Non-Confluence with ConCon 1.5 In *IWC*, 2017.

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

*Supported by Austrian Science Fund (FWF), projects P27502, and P27528.

CO3 (Version 1.4)

Naoki Nishida[1], Yoshiaki Kanazawa[1], and Karl Gmeiner[2]

[1] Nagoya University, Nagoya, Japan
{nishida@, yoshiaki@trs.css.}i.nagoya-u.ac.jp
[2] UAS Technikum Wien, Vienna, Austria
gmeiner@technikum-wien.at

CO3, a converter for proving confluence of conditional TRSs, tries to prove confluence of conditional term rewriting systems (CTRS) by using a transformational approach. The tool is based on the result in [3, 7, 6]: the tool first transforms a given weakly-left-linear (WLL) and ultra-WLL 3-DCTRS into an unconditional term rewriting system (TRS) by using the *SR transformation* $\mathbb{SR}$ [9, 10, 5] or the *unraveling* $\mathbb{U}$ [4, 8], and then verifies confluence of the transformed TRS. This tool is basically a converter of CTRSs to TRSs. The main expected use of this tool is the collaboration with other tools for proving confluence of TRSs, and thus this tool has very simple and lightweight functions to verify properties such as confluence and termination of TRSs. The tool is available from http://www.trs.css.i.nagoya-u.ac.jp/co3/.

The main technique for proving confluence is based on the following theorems: (a) a normal 1-CTRS $\mathcal{R}$ is confluent if $\mathcal{R}$ is WLL and $\mathbb{U}(\mathcal{R})$ is confluent [1, 3], (b) a 3-DCTRS $\mathcal{R}$ is confluent if $\mathcal{R}$ is WLL and $\mathbb{U}(\mathcal{R})$ is confluent [2, 3], (c) a WLL normal 1-CTRS $\mathcal{R}$ is confluent if $\mathbb{SR}(\mathcal{R})$ is confluent [7], and (d) a WLL and ultra-WLL 3-DCTRS $\mathcal{R}$ is confluent if $\mathbb{SR}(\mathcal{R})$ is confluent [6]. Version 1.3 tried to prove confluence of $\mathbb{SR}(\mathcal{R})$ if $\mathcal{R}$ is a 3-DCTRS, but this latest version does not try it because $\mathbb{SR}(\mathcal{R})$ is not confluent due to some auxiliary rules (see [6]).

References

[1] K. Gmeiner, B. Gramlich, and F. Schernhammer. On (un)soundness of unravelings. In *Proc. RTA 2010*, volume 6 of *LIPIcs*, pp. 119–134. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010.

[2] K. Gmeiner, B. Gramlich, and F. Schernhammer. On soundness conditions for unraveling deterministic conditional rewrite systems. In *Proc. RTA 2012*, volume 15 of *LIPIcs*, pp. 193–208. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

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

[4] M. Marchiori. Unravelings and ultra-properties. In *Proc. ALP 1996*, volume 1139 of *LNCS*, pp. 107–121. Springer, 1996.

[5] R. Nakayama, N. Nishida, and M. Sakai. Sound structure-preserving transformation for ultra-weakly-left-linear deterministic conditional term rewriting systems. In *Proc. WPTE 2016*, volume 235 of *EPTCS*, pp. 62–77. Open Publishing Association, 2017.

[6] N. Nishida. Notes on confluence of ultra-WLL SDCTRSs via a structure-preserving transformation. In *Proc. IWC 2016*, 2016. to appear.

[7] N. Nishida, M. Yanagisawa, and K. Gmeiner. On proving confluence of conditional term rewriting systems via the computationally equivalent transformation. In *Proc. IWC 2014*, pp. 24–28, 2014.

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

[9] T.-F. Şerbănuţă and G. Roşu. Computationally equivalent elimination of conditions. In *Proc. RTA 2006*, volume 4098 of *LNCS*, pp. 19–34. Springer, 2006.

[10] T.-F. Şerbănuţă and G. Roşu. Computationally equivalent elimination of conditions. Technical Report UIUCDCS-R-2006-2693, Department of Computer Science, University of Illinois at Urbana-Champaign, 2006.

CoLL-Saigawa: A Joint Confluence Tool*

Nao Hirokawa and Kiraku Shintani

JAIST, Japan

CoLL-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/

The typical usage is: `collsaigawa <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.

CoLL-Saigawa is a joint confluence tool of CoLL v1.1 [8] and Saigawa v1.8 [4]. If an input TRS is left-linear, CoLL proves confluence. Otherwise, Saigawa analyzes confluence. CoLL is a confluence tool specialized for left-linear TRSs. It proves confluence by using Hindley's commutation theorem [3] together with the three commutation criteria: Development closeness [2, 9], rule labeling with weight function [10, 1], and Church-Rosser modulo A/C [6]. Saigawa can deal with non-left-linear TRSs. The tool employs the four confluence criteria: The criteria based on critical pair systems [5, Theorem 3] and on extended critical pairs [7, Theorem 2], rule labeling [10], and Church-Rosser modulo AC [6]. Saigawa uses $\mathsf{T_TT_2}$ and MU-TERM to check (relative) termination.[1] A suitable rule labeling is searched by using MiniSmt.[2]

This version of CoLL-Saigawa is still at the experimental stage. Full integration of the two tools is planned for the next version.

References

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

[2] 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.

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

[4] N. Hirokawa. Saigawa: A confluence tool. In *3rd Confluence Competition (CoCo 2014)*, pages 1–1, 2014.

[5] N. Hirokawa and A. Middeldorp. Commutation via relative termination. In *Proc. 2nd IWC*, pages 29–33, 2013.

[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] K. Shintani and N. Hirokawa. CoLL: A confluence tool for left-linear term rewrite systems. In *Proc. 25th CADE*, LNAI, 2015. To appear.

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

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

*Partly supported by JSPS KAKENHI Grant Number 17K00011 and the JSPS Core-to-Core Program.

[1] http://colo6-c703.uibk.ac.at/ttt2/ and http://zenon.dsic.upv.es/muterm/

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

CoCo 2017 Participant: ConCon 1.5*

Thomas Sternagel and Christian Sternagel and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria
{thomas.sternagel, christian.sternagel, aart.middeldorp}@uibk.ac.at

ConCon is a fully automatic confluence checker for *oriented* first-order conditional term rewrite systems (CTRSs). It is written in Scala and available under the LGPL license. You can use its web interface or download it at

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

For some of its methods ConCon issues calls to the external unconditional confluence and termination checkers CSI and $\mathsf{T_TT_2}$ as well as the theorem prover Waldmeister. For a full system description of an earlier version of ConCon see [4]. ConCon first tries to simplify rules and remove infeasible rules from the input system, then it employs the following three confluence criteria:

(A) a quasi-decreasing strongly deterministic 3-CTRS is confluent if all its critical pairs are joinable [1],

(B) an almost orthogonal extended properly oriented right-stable 3-CTRS is confluent [7],

(C) a deterministic 3-CTRS $\mathcal{R}$ is confluent if its unraveling $\mathsf{U}(\mathcal{R})$ is left-linear and confluent [8].

In parallel ConCon also tries to show non-confluence using conditional narrowing (and some other heuristics). To make criteria (A) and (B) more useful, ConCon uses a variety of methods to check for infeasibility [5] of conditional critical pairs, ranging from a simple technique based on unification, via symbol transition graph analysis, reachability problem decomposition, the exploitation of certain equalities in the conditions, and tree automata completion to equational reasoning. ConCon can generate certifiable output for all implemented methods [2,3,6,8] except the infeasibility check employing equational reasoning via Waldmeister.

References

[1] J. Avenhaus and C. Loría-Sáenz. On Conditional Rewrite Systems with Extra Variables and Deterministic Logic Programs. In *Proc. 5th LPAR*, volume 822 of *LNAI*, pages 215–229, 1994.

[2] C. Sternagel and T. Sternagel. Certifying Confluence of Almost Orthogonal CTRSs via Exact Tree Automata Completion. In *Proc. 1st FSCD*, volume 52 of *LIPIcs*, pages 29:1–29:16, 2016.

[3] C. Sternagel and T. Sternagel. Certifying Confluence of Quasi-Decreasing Strongly Deterministic Conditional Term Rewrite Systems. In *Proc. 26th CADE*, volume 10395 of *LNCS*, pages 413–431, 2017.

[4] T. Sternagel and A. Middeldorp. Conditional Confluence (System Description). In *Proc. Joint 25th RTA and 12th TLCA*, volume 8560 of *LNCS*, pages 456–465, 2014.

[5] T. Sternagel and A. Middeldorp. Infeasible Conditional Critical Pairs. In *Proc. 4th IWC*, pages 13–17, 2015.

[6] T. Sternagel and C. Sternagel. Certified Non-Confluence with ConCon 1.5. This volume, 2017.

[7] T. Suzuki, A. Middeldorp, and T. Ida. Level-Confluence of Conditional Rewrite Systems with Extra Variables in Right-hand Sides. In *Proc. 6th RTA*, volume 914 of *LNCS*, pages 179–193, 1995.

[8] R. Thiemann and S. Winkler. Formalizing soundness and completeness of unravelings. In *Proc. 10th FroCoS*, volume 9322 of *LNCS*, pages 239–255, 2015.

*Supported by FWF (Austrian Science Fund) project P27502.

CoCo 2017 Participant: CSI 1.1*

Bertram Felgenhauer, Julian Nagele, and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria

CSI is a strong automatic tool for (dis)proving confluence of first-order term rewrite systems (TRSs). It is based on the termination prover TTT2 [4] and has been in development since 2010. Its name is derived from the Confluence of the rivers Sill and Inn in Innsbruck. The tool is available from

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

under a LGPLv3 license. A new improved web interface is available as well. Below we briefly report on recent extensions that make CSI more powerful, secure, and useful. A more detailed description can be found in [5].

TRSs that contain AC rules pose a challenge for confluence provers. In CSI we incorporated a version of the AC critical pair lemma based on extended rules [7], which is used in the modern completion tool mkbtt [8]. For unique normal form properties, we now support Chew's theorem [1] for UNC and, for ground TRSs, a decision procedure for NFP (in addition to CR, UNC and UNR [2, 3]). The most recent addition to CSI's repertoire of certifiable confluence criteria is based on terminating critical-pair-closing systems [6]. The following table demonstrates the progress CSI has made in the last 6 years; CSI 0.1 was released in 2011, CSI 0.6 participated in CoCo 2016. The results in the final column are using CSI's certified mode.

	CSI 0.1	CSI 0.6	CSI 1.1	✓CSI 1.1
yes	116	181	215	119
no	51	62	67	67
maybe	142	66	27	123

References

[1] P. Chew. Unique normal forms in term rewriting systems with repeated variables. In *Proc. 13th STOC*, pages 7–18, 1981. doi: 10.1145/800076.802452.

[2] B. Felgenhauer. Deciding confluence of ground term rewrite systems in cubic time. In *Proc. 23rd RTA*, volume 15 of *LIPIcs*, pages 165–175, 2012. doi: 10.4230/LIPIcs.RTA.2012.165.

[3] B. Felgenhauer. Efficiently deciding uniqueness of normal forms and unique normalization for ground TRSs. In *Proc. 5th IWC*, pages 16–20, 2016.

[4] 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. doi: 10.1007/978-3-642-02348-4_21.

[5] J. Nagele, B. Felgenhauer, and A. Middeldorp. CSI: New evidence – a progress report. In *Proc. 26th CADE*, volume 10395 of *LNCS (LNAI)*, pages 385–397, 2017. doi: 10.1007/978-3-319-63046-5_24.

[6] M. Oyamaguchi and N. Hirokawa. Confluence and critical-pair-closing systems. In *Proc. 3rd IWC*, pages 29–33, 2014.

[7] G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *JACM*, 28(2):233–264, 1981. doi: 10.1145/322248.322251.

[8] S. Winkler and A. Middeldorp. Normalized completion revisited. In *Proc. 24th RTA*, volume 21 of *LIPIcs*, pages 319–334, 2013. doi: 10.4230/LIPIcs.RTA.2013.319.

*Supported by FWF (Austrian Science Fund) project P27528.

CoCo 2017 Participant: CSIˆho 0.3*

Julian Nagele

Department of Computer Science, University of Innsbruck, Austria
julian.nagele@uibk.ac.at

Higher-order rewriting combines first-order rewriting with notions and concepts from λ-calculus, resulting in rewrite systems with higher-order functions and bound variables. CSIˆho is a tool for automatically proving confluence of such higher-order systems, specifically pattern rewrite systems (PRSs) as introduced by Nipkow [3, 6]. The restriction to pattern left-hand sides is essential for obtaining decidability of unification and thus makes it possible to compute critical pairs. To this end CSIˆho implements a version of Nipkow's algorithm for higher-order pattern unification [7]. CSIˆho is an extension of CSI, a powerful confluence prover for first-order term rewrite systems. The tool and a web interface are available from

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

Below we briefly describe the criteria implemented by CSIˆho, a more detailed description (also of CSI) can be found in [5].

The first criterion is based on a higher-order version of the critical pair lemma, that is, for terminating PRSs we decide confluence by checking joinability of critical pairs [6]. For showing termination CSIˆho implements a basic higher-order recursive path ordering and static dependency pairs with dependency graph decomposition and the subterm criterion. Alternatively, one can also use an external termination tool like WANDA [2] as an oracle. For potentially non-terminating systems CSIˆho supports two more classical criteria based on critical pairs, namely weak orthogonality [9] and van Oostrom's result on development closed critical pairs [8]. As a divide-and-conquer technique CSIˆho implements modularity, i.e., decomposing a PRS into parts with disjoint signatures, for left-linear PRSs—note that confluence of PRSs is not modular in general [1]. Finally CSIˆho uses the simple technique of adding and removing redundant rules [4], adapted for PRSs, e.g. for finding non-joinable peaks via forward closures.

References

[1] C. Appel, V. van Oostrom, and J. G. Simonsen. Higher-order (non-)modularity. In *Proc. 21st RTA*, volume 6 of *LIPIcs*, pages 17–32, 2010.

[2] Cynthia Kop. *Higher Order Termination*. PhD thesis, Vrije Universiteit, Amsterdam, 2012.

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

[4] J. Nagele, B. Felgenhauer, and A. Middeldorp. Improving automatic confluence analysis of rewrite systems by redundant rules. In *Proc. 26th RTA*, volume 36 of *LIPIcs*, pages 257–268, 2015.

[5] J. Nagele, B. Felgenhauer, and A. Middeldorp. CSI: New evidence – a progress report. In *Proc. 26th CADE*, volume 10395 of *LNCS (LNAI)*, pages 385–397, 2017.

[6] T. Nipkow. Higher-order critical pairs. In *Proc. 6th LICS*, pages 342–349, 1991.

[7] Tobias Nipkow. Functional unification of higher-order patterns. In *Proc. 8th LICS*, pages 64–74, 1993.

[8] V. van Oostrom. Developing developments. *TCS*, 175(1):159–181, 1997.

[9] V. van Oostrom and F. van Raamsdonk. Weak orthogonality implies confluence: The higher order case. In *Proc. 3rd LFCS*, volume 813 of *LNCS*, pages 379–392, 1994.

*Supported by Austrian Science Fund (FWF), project P27528.

CoCo 2017 Participant: FORT 1.0*

Franziska Rapp and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria
{franziska.rapp|aart.middeldorp}@uibk.ac.at

FORT is a decision and synthesis tool for the first-order theory of rewriting for finite left-linear right-ground rewrite systems. It implements the decision procedure for this theory, which uses tree automata techniques and goes back to Dauchet and Tison [1]. In this theory confluence-related properties on ground terms are easily expressible. The basic functionality of FORT is described in [2] and in [3] we report on an extension to deal with non-ground terms.

FORT 1.0 was implemented 2016 in Java, for which the JAR file can be downloaded from

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

The tool participates in the categories UNC, NFP, UNR and GCR at CoCo 2017. The latter is about ground-confluence of *many-sorted* rewrite systems. Since the set of well-typed terms according to a many-sorted type discipline is accepted by a tree automaton, the modifications required in FORT were straightforward.

The most significant change in FORT 1.0 is the support for parallelism, using the multi-threading capabilities of Java. This greatly speeds up the synthesis of rewrite systems satisfying certain properties expressible in the first-order theory of rewriting. Furthermore, we exploit this functionality for deciding properties.

References

[1] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th IEEE Symposium on Logic in Computer Science*, pages 242–248, 1990. doi: 10.1109/LICS.1990.113750.

[2] F. Rapp and A. Middeldorp. Automating the first-order theory of left-linear right-ground term rewrite systems. In *Proc. 1st International Conference on Formal Structures for Computation and Deduction*, volume 52 of *Leibniz International Proceedings in Informatics*, pages 36:1–36:12, 2016. doi: 10.4230/LIPIcs.FSCD.2016.36.

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

*Supported by FWF (Austrian Science Fund) project P27528.

The System SOL: Second-Order Laboratory

Makoto Hamana[1], Tatsuya Abe[2], Yuito Murase[3], Kazuhiko Sakaguchi[4]

[1] Department of Computer Science, Gunma University, Japan
hamana@cs.gunma-u.ac.jp
[2] STAIR Lab, Chiba Institute of Technology, Japan
abet@stair.center
[3] Department of Computer Science, University of Tokyo, Japan
murase@lyon.is.s.u-tokyo.ac.jp
[4] Department of Computer Science, University of Tsukuba, Japan
murase@lyon.is.s.u-tokyo.ac.jp

SOL is a Haskell-based tool that assists the proofs of confluence and strong normalisation of higher-order computation. SOL is intended to be a tool that is applicable to the modern theories of higher-order programming languages. This aim is demonstrated in the first author's recent article [Ham17b], which is presented at ICFP'17.

The system SOL supports multiple formats as input, including, CoCo's HRS format, XML higher-order rule format in TPDB, and SOL's original format. The last format is written in an embedded domain specific language using the feature of Template Haskell.

Based on the foundation of second-order algebraic theories [FH10] and its computational counter part [Ham16, Ham17b], we implemented various results on higher-order syntax and computation in SOL, including Knuth and Bendix's criterion for confluence, deterministic second-order pattern matching [YHT04] for matching, and Function-as-Constructor Unification (FCU) [LM16] for unification. We implemented it by extending Nipkow's functional implementation of higher-order pattern unification [Nip93], which we report in [Ham17a]. Termination analysis is based on the General Schema criterion [Bla00].

References

[Bla00] F. Blanqui. Termination and confluence of higher-order rewrite systems. In *Rewriting Techniques and Application (RTA 2000)*, LNCS 1833, pages 47–61. Springer, 2000.

[FH10] M. Fiore and C.-K. Hur. Second-order equational logic. In *Proc. of CSL'10*, LNCS 6247, pages 320–335, 2010.

[Ham16] M. Hamana. Strongly normalising cyclic data computation by iteration categories of second-order algebraic theories. In *Proc. of FSCD'16*, volume 52 of *the Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:18, 2016.

[Ham17a] M. Hamana. A functional implementation of function-as-constructor higher-order unification. *Proc. 31st International Workshop on Unification (UNIF'17)*, 2017. to appear.

[Ham17b] M. Hamana. How to prove your calculus is decidable: practical applications of second-order algebraic theories and computation. *Proceedings of the ACM on Programming Languages*, 1(1):22:1–22:28, 9 2017. to appear in ICFP'17.

[LM16] T. Libal and D. Miller. Functions-as-Constructors Higher-Order Unification. In *Proc. of FSCD 2016*, volume 52 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:17, 2016.

[Nip93] T. Nipkow. Functional unification of higher-order patterns. In *Proc. of (LICS'93)*, pages 64–74, 1993.

[YHT04] T. Yokoyama, Z. Hu, and M. Takeichi. Deterministic second-order patterns. *Inf. Process. Lett.*, 89(6):309–314, 2004.