Proceedings of the

9th International Workshop on Confluence

June 30th 2020

Online

Foreword

This report contains the proceedings of the 9th International Workshop on Confluence (IWC) which took place on June 30th, 2020. It was due to be held in Paris, but had to be changed to a fully online event due to the coronvirus pandemia. In addition, the proceedings include the system descriptions of the 9th Confluence Competition (CoCo 2020). The workshop was part of the Paris Nord Summer of LoVe 2020, a joint event on LOgic and VErification at Université Paris 13, made of Petri Nets 2020, IJCAR 2020, FSCD 2020, and over 20 satellite events. We would like to thank the conference chair Stefano Guerrini and the workshop organizer Giulio Manzonetto for their hard work on entirely changing the organization of the conference.

Confluence provides a general notion of determinism and has been conceived as one of the central properties of rewriting systems. Confluence relates to many topics of rewriting (completion, modularity, termination, commutation, etc.) and has been investigated in many formalisms of rewriting, such as first-order rewriting, lambda-calculi, higher-order rewriting, constraint rewriting, conditional rewriting, and so on. Recently there is a renewed interest in confluence research, resulting in new techniques, tool support, confluence competition, and certification as well as in new applications. The scope of the workshop is all these aspects of confluence and related topics. The goal of the IWC workshop is to provide a forum for researchers interested in the topic of confluence to exchange and share new developments in the field. The workshop will enable discussion on theoretical results, new problems, applications, implementations and benchmarks, and share the current state-of-the-art on the development of confluence tools.

The joint program contains 8 contributed talks as well as invited talks by Frédéric Blanqui and Margherita Zorzi. In addition, the program contains the system descriptions from the 8th Confluence Competition (CoCo 2020). Many people contributed to the preparation and IWC. Hard work by the program commitees, steering committees, and subreviewers made an exciting program of contributed and invited talks possible. In addition, we are greatful to the organizing committee and workshop chairs of FSCD-IJCAR for hosting the workshops.

June 11th, 2020, Paris
Mauricio Ayala-Rincón
Samuel Mimram

Steering Committee

- Takahito Aoto
- Nao Hirokawa

Program Committee

- Beniamino Accattoli (INRIA & LIX, École Polytechnique)
- Mauricio Ayala-Rincón (Universidade de Brasília), co-chair
- Cyrille Chenavier (Centre Inria Lille)
- Alejandro Díaz-Caro (Universidad Nacional de Quilmes & ICC/UBA-CONICET)
- Maribel Fernández (King's College London)
- Mario Florido (Universidade de Porto)
- Makoto Hamana (Gunma University)
- Philippe Malbos (Université Claude Bernard Lyon 1)
- Samuel Mimram (LIX, École Polytechnique), co-chair
- Camilo Rocha (Pontificia Universidad Javeriana - Cali)
- Daniel Lima Ventura (Universidade Federal de Goiás)
- Femke van Raamsdonk (VU University Amsterdam)
- Johannes Waldmann (Hochschule für Technik, Wirtschaft und Kultur Leipzig)
- Sarah Winkler (Università degli studi di Verona)

Additional reviewers

- Claudia Faggian
- Pablo Barenbaum

Contents

Some symmetries of commutation diamonds

Vincent van Oostrom

University of Innsbruck
Vincent.van-Oostrom@uibk.ac.at

Abstract

We study commutation of rewrite systems under swapping, reversing.

1 Introduction

Commutation may be used as a building block in various settings ranging from the abstract to the concrete. For instance, causality may be analysed as the independence of events which may be modeled as their commutation, and correctness of compiler optimisations may be modeled as commutation between evaluation and optimisation. Commutation generalises confluence, which itself may be used for establishing consistency of and deciding equational theories.

In this short paper we investigate commutation under *swapping* and *reversing* of the rewrite systems. We show that at the level of abstract rewriting this perspective, although bringing nothing new *per se*, may serve to bring unity into disparate results in the literature. At the level of term rewriting some fun is to be had, albeit simple, because in the commutation-by-critical-pair-analysis (due to Knuth and Bendix and Huet for first-order term rewriting, and extended to higher-order term rewriting by Nipkow), rules are commonly assumed to conform to certain restrictions, restrictions that may not be preserved when reversing them.

Example 1. *Reversing the term rewrite rules:*

$$f(x) \to x \qquad f(x) \to a \qquad f(x) \to g(x,x) \qquad (\lambda x.F(x))G \to F(G)$$

violates the respective requirements that: the lhs should not be a variable (x is),[1] *variables in the rhs should occur in the lhs (x does not occur in a), the lhs should be linear (x occurs twice in $g(x,x)$),*[2] *the lhs should be a pattern (in the sense of Miller; $F(G)$ is not a pattern).*

Instead of a conclusion we have **positions** after each subtopic.[3]

2 Diamond symmetries in abstract rewriting

We assume knowlege of abstract and term rewriting [7]. We employ arrow-like notations $\blacktriangleright$, $\triangleright$, ... for abstract rewrite systems or relations on a (the same) set of objects, reversing these notations $\blacktriangleleft$, $\triangleleft$, ... to denote the reverse systems,and their repetitions $\blacktriangleright\!\!\blacktriangleright$, $\triangleright\!\!\triangleright$, ... to denote the reflexive–transitive closures. We define $\blacktriangleleft\!\triangleright := \blacktriangleleft \cup \triangleright$ and $\to := \blacktriangleright \cup \triangleright$. As in [5], we extend terminology and notation for a single rewrite system $\blacktriangleright$ to the *diagonal* $\blacktriangleright, \blacktriangleright$, i.e. such that $P(\blacktriangleright)$ iff $P(\blacktriangleright, \blacktriangleright)$ for P a property of (pairs of) rewrite systems.

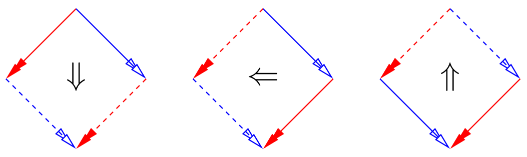
Definition 1. *The pair $\blacktriangleright$,$\triangleright$ has the* diamond *property if $\blacktriangleleft \cdot \triangleright \subseteq \triangleright \cdot \blacktriangleleft$, it* commutes *if $\blacktriangleright\!\!\blacktriangleright$,$\triangleright\!\!\triangleright$ has the diamond property, and has the* Church–Rosser *property if $(\blacktriangleleft \cup \triangleright)^* \subseteq \triangleright\!\!\triangleright \cdot \blacktriangleleft\!\!\blacktriangleleft$.*

[1] Variable-lhss are allowed by Huet in his Critical Pair Lemma, but not in Nipkow's higher-order version.

[2] Although non-left-linear rules are usually allowed in term rewriting, they do not go well with commutation: Hirokawa and Shintani show in IWC 2015, for them commutation is not even preserved by signature extension.

[3] For lack of space we often only provide hyperlinks for informal references to the literature.

We may refer to the diagonal of P as *self-*P. For instance, self-commutation is conventionally known as *confluence*. The name of the diamond property derives from that it can be visualised like that. The 8 symmetries of such a diamond (the dihedral group D_4) give rise to 8 instances of the diamond property, obtained by *swapping* and *reversing* the rewrite systems $\blacktriangleright, \triangleright$, and hence also of commutation. We have visualised[4] three of these, together with their respective (conventional) names:[5]



commutation factorisation upward commutation

Formally, letting P range over the properties in Definition 1, left–right *reflecting* the diamond corresponds to *swapping* the pair $\blacktriangleright, \triangleright$ into $\triangleright, \blacktriangleright$ and is a symmetry: $P(\blacktriangleright, \triangleright)$ iff $P(\triangleright, \blacktriangleright)$.[6] Clockwise *rotating* the diamond corresponds to first *swapping* the pair $\blacktriangleright, \triangleright$ and then *reversing* the first element yielding $\triangleleft, \blacktriangleright$. The first displayed rotation turns commutation into *factorisation*, $\triangleright\!\!\triangleright \cdot \blacktriangleright\!\!\blacktriangleright \subseteq \blacktriangleright\!\!\blacktriangleright \cdot \triangleright\!\!\triangleright$, which is equivalent to the Church–Rosser property of $\triangleleft, \blacktriangleright$, i.e. that a reduction $a \twoheadrightarrow b$ *factors* as $a \blacktriangleright\!\!\blacktriangleright \cdot \triangleright\!\!\triangleright\, b$. Symmetries preserve results. E.g. that the diamond property implies the Church–Rosser property is the same result as that $\triangleright \cdot \blacktriangleright \subseteq \blacktriangleright \cdot \triangleright$ implies factorisation.

Position 1. *Results for diamond and commutation are to be taken up to symmetries.*

How to show the Church–Rosser property of $\blacktriangleright, \triangleright$? Based on a note from 1978 by De Bruijn, we introduced the *decreasing diagrams* (DD) technique in my PhD thesis, requiring for each *local*[7] *peak* $a \blacktriangleleft \cdot \triangleright b$ existence of a suitably constrained *valley* $a \triangleright\!\!\triangleright \cdot \blacktriangleleft\!\!\blacktriangleleft b$, a result we extended in 2008 to allow *conversions* instead of valleys:

Definition 2. $\blacktriangleright, \triangleright$ *is* decreasing, *if* $\blacktriangleright := \bigcup_{i \in I} \blacktriangleright_i$, $\triangleright := \bigcup_{j \in J} \triangleright_j$ *for families* $(\blacktriangleright_i)_{i \in I}, (\triangleright_j)_{j \in J}$ *and some well-founded strict order* $<$ *on* $I \cup J$, *such that for all* $i \in I, j \in J$ ${}_i\!\blacktriangleleft \cdot \triangleright_j \subseteq \blacktriangleleft\!\triangleright^*_{\curlyvee i} \cdot \triangleright_j^= \cdot \blacktriangleleft\!\triangleright^*_{\curlyvee\{i,j\}} \cdot {}_i\!\blacktriangleleft^= \cdot \blacktriangleleft\!\triangleright^*_{\curlyvee j}$, *where* $\curlyvee K := \{k \in I \cup J \mid \exists \ell \in K\ \ell > k\}$ *and* $\curlyvee k := \curlyvee\{k\}$.

Theorem 1 (Decreasing Diagrams, DD [4])**.** $\blacktriangleright, \triangleright$ *commute if decreasing.*

Since then, DD has found wide application in the literature in the study of confluence and commutation (see below), but somewhat surprisingly (given their symmetry), as far as we know, not yet within the study of factorisation. Here and in Section 3 we give examples[8] illustrating the power of DD also for establishing factorisation results, and at the same time how DD allows one to focus on extracting appropriate families and orders on them, easing applicability.

Example 2. $\blacktriangleright := \bigcup_i \blacktriangleright_{i \in I}$ *and* $\triangleright := \bigcup_{j \in I} \triangleright_j$ *commute if for all* i, j, $\triangleright_j$ *is terminating and*

$$ {}_i\!\blacktriangleleft \cdot \triangleright_j \subseteq \triangleright_j \cdot ({}_i\!\blacktriangleleft \cup \triangleright_i)^* \qquad (1)$$

To see this, first note it suffices to show $\blacktriangleright_i, \triangleright$ *commute for all* i. *Fixing* i *allows to turn* (1) *into a DD by decomposing* $\triangleright$ *into* $\triangleright_i$ *and* $\triangleright_{\neg i} := \bigcup_{k \neq i} \triangleright_k$, *ordering* $\triangleright_{\neg i}$*-steps above others, and ordering* $\blacktriangleright_i$*- and* $\triangleright_i$*-steps via their targets by* $(\blacktriangleright\!\!\blacktriangleright_i \cdot {}_i\!\triangleleft \cdot \blacktriangleright\!\!\blacktriangleright_i)^+$, *well-founded by* (1) *[7, Exc. 1.3.19].*

[4]As usual, ordinary/dashed arrows represent universally/existentially quantified steps and reductions.

[5]Other names of factorisation are *postponement* (from a $\triangleright$-perspective) or *preponement* ($\blacktriangleright$-perspective).

[6]This is apparent via left–right reflection of our formal notations, as these mirror those of the diamond.

[7]Following Newman's 1942 in *localising* properties P, we use *local* P to refer to P with its assumption restricted to single steps ($\blacktriangleright$,$\triangleright$) instead of general reductions ($\blacktriangleright\!\!\blacktriangleright$,$\triangleright\!\!\triangleright$). E.g., *local* commutation is $\blacktriangleleft \cdot \triangleright \subseteq \triangleright\!\!\triangleright \cdot \blacktriangleleft\!\!\blacktriangleleft$.

[8]Found by us over the past 20 years, but only privately communicated and circulated.

That families $\to_\bullet := (\rightsquigarrow_\bullet, \rightarrowtail_\bullet)$ and $\to_\circ := (\rightsquigarrow_\circ, \rightarrowtail_\circ)$ satisfy the conditions of a square factorisation system *[1], directly entails $\rightsquigarrow_\circ, \rightarrowtail_\circ$ are terminating and (1) holds for*[9] ${}_\bullet\!\leftarrow, \to_\circ$*, yielding the main abstract factorisation result of that paper [1, Thm. 5.2].*

Note that using DD not only allowed our statement and proof to be (much) more compact,[10] but also our result to be (much) more general, not just because of allowing arbitrary size families; already for families of size 2 it is more general: e.g. where square factorisation systems require $\rightsquigarrow_\bullet \cdot \rightsquigarrow_\circ$ to be contained in $\rightsquigarrow_\circ^+ \cdot \rightsquigarrow_\bullet^+$, (1) only requires it to be contained in $\rightsquigarrow_\circ \cdot (\rightsquigarrow_\bullet \cup \rightsquigarrow_\circ)^+$ and similarly for $\rightarrowtail$. In [4] we showed that in fact *all* 'local commutation ⇒ commutation' results we knew of then, could be obtained as instances of DD. We did so by introducing various basic techniques for finding families and orders such as *self*- and *rule*-labelling. For instance, the Lemma of Hindley–Rosen was obtained by taking for $>$ the *empty* order on families [4, Example 13], and the commutation version (due to Backhouse and Doornbos) of Newman's Lemma was obtained by *self*-labelling a step $a \to b$ by its *source* and ordering labels by $\leftarrow^+$ [4, Example 12].[11] DD has been formalised in Isabelle and is part of the AFP, due to Zankl for Theorem 1 and to Felgenhauer for a *proof order* version. DD has been automated in tools such as ACP and CSI, and many commutation results have been *factored through* DD and often generalised by it, e.g. Toyama's famous modularity of confluence result; see the introduction of [2] for more examples. Still we think that (much) more leverage could be gotten out of DD.

Remark 1. *A main feature of the TRS confluence tool CoLL [6] is that it is not built on* confluence *but on* commutation *criteria. The idea is to (rule-based) decompose a TRS $\mathcal{R}$ into a family $(\mathcal{R}_i)_i$ of TRSs. Confluence of $\mathcal{R}$ follows by the Lemma of Hindley–Rosen from commutation of all pairs (including the diagonal) $\mathcal{R}_i, \mathcal{R}_j$ of family members. We suggest also there employing DD, instead of the lemma of Hindley–Rosen, could be beneficial. In fact, an example of such a decomposition based on DD was already given as [4, Thm. 5]. Whether/how such decompositions could be found automatically remains to be investigated. Moreover, it seems interesting to consider decompositions other than rule-based ones, e.g. ones obtained by instantiation or by strategies.*

Position 2. *DD is the swiss-army-knife for commutation, its symmetries (factorisation), and its instances (confluence). Trying to extract appropriate families and orders on them for establishing these properties via DD, results often in (more) powerful yet (more) compact results.*

How powerful is the DD technique exactly, for showing confluence and commutation?

Theorem 2.[12] *The DD technique is* complete *for confluence of rewrite systems. More precisely, every countable*[13] *confluent rewrite system $\to$, is decreasing for some family and order.*

As the name suggests, more than being a 1-dimensional confluence result, DD establishes a 2-*dimensional* diagrammatic confluence result: Every peak $b \twoheadleftarrow a \twoheadrightarrow c$ *can* be completed by some valley $b \twoheadrightarrow d \twoheadleftarrow c$ into a confluence *diagram*, by means of repeatedly adjoining *locally* decreasing *diagrams*; even stronger, repeatedly adjoining such locally decreasing diagrams *must* terminate after finitely many steps into a decreasing confluence diagram. We now show that countable

[9]Note the reversal for the relations of the first family, turning commutation into factorisation.

[10]2 lines vs. 8 lines respectively 3 lines vs. 3 pages.

[11]Pous showed in 2005 that termination of $(◀^+ \cdot ◁^+)^+$ instead of of $\leftarrow^+$ suffices; again DD applies, but using *step* labelling instead of source-labelling [4, Example 17].

[12]This was established independently by Ken Mano and the author (see Remark 2.3.29 in my PhD thesis). The proofs employed a decomposition into a natural–number-indexed family. Recently Klop asked the question whether smaller families suffice, upon which Endrullis, Klop and Overbeek showed that in fact doubletons do.

[13]The case of uncountable systems, conjectured to be false in my PhD thesis, remains open.

confluence suffices even to obtain a 3-*dimensional* (decreasing) diagrammatic confluence result, by a process we dub *cutting*[14] *faces*, which will be discussed more extensively below.[15]

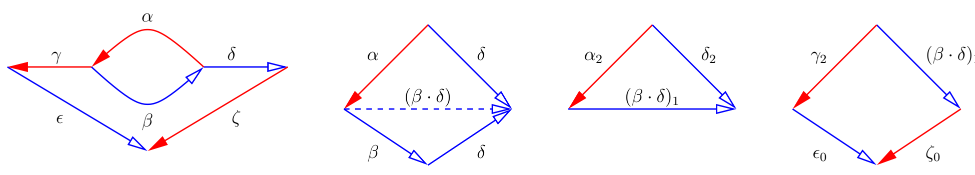
Lemma 1.[16] *Every countable confluent rewrite system $\to$ admits a residual system [7, Sect. 8.7] $(\mathrel{-\!\!\ominus\!\!\to}, 1, /)$ with $\to \subseteq \mathrel{-\!\!\ominus\!\!\to} \subseteq \twoheadrightarrow$.*

Proof. By countable confluence $\to$ has a *spanning forest* F [2, Lem. 1]. Let the relation $\mathrel{-\!\!\ominus\!\!\to}$ comprise the *steps* of $\to$ not in F, and the *reductions* of F. Viewing steps of an F-reduction as its *faces* its transformation into a single $\mathrel{-\!\!\ominus\!\!\to}$-step can be viewed as *cutting* faces, in this case resulting in a *diamond*: Define the *residual* ϱ/ϕ of $\varrho : a \mathrel{-\!\!\ominus\!\!\to} b$ *after* $\phi : a \mathrel{-\!\!\ominus\!\!\to} c$ by cases on their sources and targets to be ϱ if $c = a$, 1 if $a = b$, and $c \mathrel{-\!\!\ominus\!\!\to} d$ otherwise with d the *least common descendant* of b, c in F. One checks that the (designated) diamond property, i.e. if $\varrho : a \mathrel{-\!\!\ominus\!\!\to} b$, $\phi : a \mathrel{-\!\!\ominus\!\!\to} c$ then $\varrho/\phi : c \mathrel{-\!\!\ominus\!\!\to} d$, $\phi/\varrho : b \mathrel{-\!\!\ominus\!\!\to} d$ for some d, and the (3-dimensional) *cube* identity $(\varrho/\phi)/(\psi/\phi) = (\varrho/\psi)/(\phi/\psi)$ hold by uniqueness of least common descendants in F. ☐

The process of *tiling* by 2-dimensional local diagrams, introduced in [3] and later resumed by Melliès, for the purpose of establishing confluence and standardisation, need *not* entail the 3-dimensional cube property; the edges of the 6 plane surfaces obtained by 2-dimensional tiling need *not* match up. It fails both for terms [7, Fig. 8.53] and (positive) braids [7, Sect. 8.9].

Position 3. *It is of interest to study residual systems for commutation (*coloured *cubes).*

Is DD also complete for *commutation*? We raised this question in [4] and at ISR 2008. It was summarily confuted, then and there, by two participants, Endrullis and Grabmayer:



The rewrite system on the left is not (and cannot be made) decreasing. Even stronger, although the system commutes, that cannot be shown by *tiling* with local commutation diagrams, as it embeds (omit ϵ, ζ) the standard counterexample against 'local commutation $\Rightarrow$ commutation'. As in the previous section, cutting faces comes to the rescue: Looking at the commutation diagram (2nd from left) for the local peak ${}_{\alpha}\blacktriangleleft \cdot \triangleright_{\delta}$ we see it is not yet a diamond since it needs the 2-step *reduction* comprising β and δ. However, *cutting* the reduction into a fresh *face* (step) we name $(\beta \cdot \delta)$, the local diagram (3rd from left) cuts a better figure/diamond.[17] Adjoining $(\beta \cdot \delta)$ and, symmetrically, $(\alpha \cdot \gamma)$ gives rise to new peaks, but we see these can also be completed into diamonds (right diagram). The rewrite system obtained is decreasing for the labels in the natural numbers ordered by $\leq$, as displayed in the rightmost two diagrams. Cutting is a common process, e.g., *parallel* steps $\mathrel{-\!\!\!+\!\!\!+\!\!\to}$ for TRSs and *multi-steps* $\mathrel{-\!\!\ominus\!\!\to}$ for HRSs and braids [7] can be seen as being obtained by repeatedly cutting diagrams into diamonds.[18]

Position 4. Minimally[19] *cutting faces is useful to get diamonds, decreasing diagrams and cubes.*

[14]Our (tentative) naming is based on that used for processing raw diamonds. However, adjoining *transitive* inferences (cutting corners), is at the basis of both our *cuts* and those in proof theory, where, e.g., $\Gamma \vdash \Delta$ may be obtained by *cutting* the occurrence of A between $\Gamma \vdash A$ and $A \vdash \Delta$.

[15]We learned the technique of cutting faces in 1995 from Hans Zantema, for braids.

[16]This result was obtained in 2008, in our collaboration with Patrick Dehornoy on the Z-property.

[17]The diamond is still not square, but could be made so by adjoining *empty* reductions reified into steps.

[18]Somewhat miracuously, in these 3 cases even cubes are obtained.

[19]Cutting reductions into steps *by need* only; if $\blacktriangleright, \triangleright$ commutes, simply taking $\blacktriangleright\!\blacktriangleright, \triangleright\!\triangleright$ as *steps* yields a diamond.

3 Diamond symmetries in term rewriting

Commutation between *term* rewrite systems is standardly reduced to an analysis of their *critical peaks*, cf. [6]. Symmetry suggests the same applies to factorisation, but then for peaks with respect to the *reverse* of the second system. Despite that standard term rewriting theory is not well-adapted to reverse rules, cf. Example 1, it can often be easily adapted, as we illustrate:

Example 3. *Factorisation holds in the untyped λ-calculus for $\blacktriangleright := \rightarrow_\beta$, $\triangleright := \rightarrow_\eta$. Note the reverse $P \rightarrow \lambda y.Py$ of the η-rule is not a higher-order pattern rule in the sense of Nipkow, as its left-hand side P is a* ***variable****. Still, we do have a (single) critical peak* à la Huet *with the β-rule: $Q := (\lambda y.(\lambda x.M)y)N \triangleright (\lambda x.M)N \blacktriangleright M[x{:=}N] =: R$; in rewriting terminology, the $\triangleright$-step is said to* create *the $\blacktriangleright$-step. Toward factorisation, note the critical peak can be completed as $Q \blacktriangleright (\lambda x.M)N \blacktriangleright R$. Observing the first $\blacktriangleright$-step is* affine *(non-duplicating) we decompose $\blacktriangleright$ into affine $\blacktriangleright_1$ and non-affine steps $\blacktriangleright_2$. We claim that then all local peaks are decreasing for the order $\blacktriangleright_1 < \triangleleft < \blacktriangleright_2$. For the above critical diagram this holds per construction of our order. A non-critical, i.e. non creating, local peak $\triangleright \cdot \blacktriangleright_i$ can be completed by $\blacktriangleright_i \cdot \triangleright\!\!\triangleright$ by standard residual theory, using that the η-rule is* ***linear****, which again yields a decreasing diagram.*

Position 5. *Extending Nipkow's higher-order critical pair lemma to allow for variable-lhss (à la Huet) is useful. Lifting results should be unproblematic (exception: development-closedness).*

Example 4. *To show* head,internal-*factorisation for untyped $\lambda\beta$-calculus, first note that although critical peaks seem intricate as the reverse of the β-rule is not a* ***pattern****-rule (Example 1), here there are in fact none as a step creating a head-step outside it must be head itself, so not internal. Next, note it suffices to show $\blacktriangleright,\triangleright$-factorisation for $\blacktriangleright := \rightarrow_h$ and $\triangleright := \mathrel{-\!\!\circ\!\!\rightarrow}_i$ since $\rightarrow_i \subseteq \mathrel{-\!\!\circ\!\!\rightarrow}_i \subseteq \twoheadrightarrow_i$. Finally, we conclude by DD ordering $\blacktriangleright < \triangleleft$ since $\triangleright \cdot \blacktriangleright \subseteq \blacktriangleright \cdot \mathrel{-\!\!\circ\!\!\rightarrow} \subseteq \blacktriangleright \cdot \blacktriangleright\!\!\blacktriangleright \cdot \triangleright^=$ where the 1st inclusion holds by Church–Rosser as the $\blacktriangleright$-step must be a* unique *residual as $\rightarrow_i$-steps can neither create (noted) nor* ***replicate*** *$\blacktriangleright$-steps, and the 2nd*[20] *by exhaustively (it stops by Finite Developments) selecting $\blacktriangleright$-steps from $\mathrel{-\!\!\circ\!\!\rightarrow}$ until the residual is a $\triangleright$-step or empty.*[21]

Position 6. *Factorisation of term rewrite system(strategie)s is best*[22] *analysed by a critical peak analysis between rules and reverse rules.*

References

[1] B. Accattoli. An abstract factorization theorem for explicit substitutions. In *Proc. 23rd RTA*, volume 15 of *LIPIcs*, pages 6–21. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2012.

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

[3] J.W. Klop. *Combinatory Reduction Systems.* PhD thesis, Rijksuniversiteit Utrecht, 1980.

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

[5] V. van Oostrom and Y. Toyama. Normalisation by Random Descent. In *Proc. 1st FSCD*, volume 52 of *LIPIcs*, pages 32:1–32:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

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

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

[20] Thanks to the IWC reviewers for pointing out that by mistake I had oversimplified this (part of the) proof.

[21] A similar analysis applies to strategies other than head, e.g. spine or left.

[22] As observed by Geuvers in (the 'Stellingen' going with) his PhD thesis, the above critical peak between the reverse of η and β, was missed by Barendregt in the proof of Corollary 15.1.5 in his book The Lambda Calculus.

Parallel Closedness Revisited

Kiraku Shintani[1] and Nao Hirokawa[1]

JAIST, Japan
{s1820017,hirokawa}@jaist.ac.jp[*]

Abstract

In this note we present a simple proof of Huet's parallel closedness theorem (1980). We also show that Toyama's almost parallel closedness (1988), a generalization of Huet's theorem, is subsumed by his earlier result (1981), which is known to be a generalization of Gramlich's confluence criterion based on parallel critical pairs (1996).

1 Introduction

The parallel closedness theorem by Huet [3] is a pioneering work in analyzing confluence of left-linear term rewrite systems (TRSs). It claims that a left-linear TRS is confluent if every critical pair is closed by parallel step $\mathrel{\;\parallel\!\!\!\!\!\rightarrow}$. His theorem, including its ingenious proof technique, had a significant influence on later confluence research. In 1996, Gramlich [2] introduced *parallel* critical pairs to make a powerful variant of Huet's theorem. While the resulting criterion enables us to use relaxed forms for the closing condition, he also showed that it is not comparable with Huet's theorem. This might give us the impression that criteria based on parallel critical pairs cannot supersede ones based on ordinary critical pairs (see e.g. [1, Section 6.5]). However, this is wrong. We will show that Toyama's earlier result [9] subsumes Huet's theorem as well as Gramlich's criterion. Actually it subsumes the almost parallel closedness theorem [10], which is Toyama's later work and known as a generalization of Huet's theorem.

In the remaining part of the note we revisit Huet's and Toyama's parallel closedness theorems. Introducing a new induction measure, we give a simple proof of the parallel closedness theorem in Section 2. In Section 3 we show that Toyama's earlier result subsumes his later result. We assume familiarity with term rewriting [1, 8].

2 Huet's Parallel Closedness

In this section we present a simple proof of Huet's parallel closedness theorem [3] with a new measure. By denoting the critical pair $t \leftarrow \cdot \xrightarrow{\epsilon} u$ by $t \leftarrow\!\rtimes\!\xrightarrow{\epsilon} u$, the theorem is stated as follows:

Definition 1 ([3])**.** *A TRS is* parallel closed *if* $\leftarrow\!\rtimes\!\xrightarrow{\epsilon} \subseteq \mathrel{\;\parallel\!\!\!\!\!\rightarrow}$.

Theorem 1 ([3])**.** *A left-linear TRS is confluent if it is parallel closed.*

We prepare notations for our new measure. Let t be a term. The size of t is denoted by $|t|$. Given a set P of positions in t, we write $|t|_P$ for the sum of $|(t|_p)|$ for all $p \in P$. Note that $|t| \geqslant |t|_P$ holds if P is a set of parallel positions of t. We define the strict order $\succ$ as the lexicographic product of the standard order $>$ on $\mathbb{N}$ and the proper superterm relation $\rhd$. We also prepare terminologies for analyzing peaks. Let $\Gamma : t \overset{P}{\mathrel{\leftarrow\!\!\!\!\!\parallel\;}} \ell\sigma \xrightarrow{\epsilon} r\sigma$ be a peak, where $\ell \to r$ is a rewrite rule. We say that Γ is *overlapping* if some position $p \in P$ is a function position in ℓ and in the case of $P = \{\epsilon\}$ the rule employed in $t \overset{P}{\mathrel{\leftarrow\!\!\!\!\!\parallel\;}} \ell\sigma$ is not a variant of $\ell \to r$. Otherwise, Γ is *non-overlapping.* We are ready for proving the theorem.

*Supported by JSPS KAKENHI Grant Number 17K00011 and Core-to-Core Program.

$$\begin{tikzcd} s \arrow[r, "{>\epsilon}", "\shortparallel"' description] \arrow[d, "{>\epsilon}"', "\shortparallel" description] \arrow[dr, phantom, "\text{I.H.}"] & u \arrow[d, dashed, "\shortparallel" description] \\ t \arrow[r, dashed, "\shortparallel" description] & \cdot \end{tikzcd}$$

(ii) non-root

$$\begin{tikzcd} s \arrow[r, "Q", "\shortparallel"' description] \arrow[d, "\epsilon"'] \arrow[dr, phantom, "\text{PML}"] & u \arrow[d, dashed, "\epsilon"] \\ t \arrow[r, dashed, "\shortparallel" description] & \cdot \end{tikzcd}$$

(iii) non-overlapping

$$\begin{tikzcd} s \arrow[r, "q"] \arrow[d, "\epsilon"'] & s' \arrow[r, "Q'", "\shortparallel"' description] \arrow[dl, dashed, "P'", "\shortparallel"' description] \arrow[dr, phantom, "\text{I.H.}"] & u \arrow[d, dashed, "\shortparallel" description] \\ t \arrow[rr, dashed, "\shortparallel" description] & & \cdot \end{tikzcd}$$

(iv) overlapping

Figure 1: Proof of Theorem 1.

Proof of Theorem 1. Let $\mathcal{R}$ be a left-linear and parallel closed TRS. It is sufficient to show that $\mathrel{-\!\!\!\shortparallel\!\!\!\to}$ has the diamond property. Let $\Gamma\colon t \overset{P}{\mathrel{\leftarrow\!\!\!\shortparallel\!\!\!-}} s \overset{Q}{\mathrel{-\!\!\!\shortparallel\!\!\!\to}} u$ be a peak. By well-founded induction on $(|t|_P + |u|_Q, s)$ with respect to $\succ$ we show $t \mathrel{-\!\!\!\shortparallel\!\!\!\to} \cdot \mathrel{\leftarrow\!\!\!\shortparallel\!\!\!-} u$. Depending on the shape of Γ, we distinguish four cases. Figure 1 illustrates cases (ii)–(iv).

(i) If P or Q is empty then $t \mathrel{-\!\!\!\shortparallel\!\!\!\to} u$ or $t \mathrel{\leftarrow\!\!\!\shortparallel\!\!\!-} u$. In either case the claim holds.

(ii) If $P \nsubseteq \{\epsilon\}$ and $Q \nsubseteq \{\epsilon\}$ then Γ is of form $f(t_1, \ldots, t_n) \mathrel{\leftarrow\!\!\!\shortparallel\!\!\!-} f(s_1, \ldots, s_n) \mathrel{-\!\!\!\shortparallel\!\!\!\to} f(u_1, \ldots, u_n)$ with $t_i \overset{P_i}{\mathrel{\leftarrow\!\!\!\shortparallel\!\!\!-}} s_i \overset{Q_i}{\mathrel{-\!\!\!\shortparallel\!\!\!\to}} u_i$ for all $1 \leqslant i \leqslant n$. Here $P_i = \{p \mid ip \in P\}$ and $Q_i = \{q \mid iq \in Q\}$. For each $i \in \{1, \ldots, n\}$, we have $|t|_P \geqslant |t_i|_{P_i}$ and $|u|_Q \geqslant |u_i|_{Q_i}$, and therefore $|t|_P + |u|_Q \geqslant |t_i|_{P_i} + |u_i|_{Q_i}$. Since $(|t|_P + |u|_Q, s) \succ (|t_i|_{P_i} + |u_i|_{Q_i}, s_i)$ holds, the induction hypothesis yields $t_i \mathrel{-\!\!\!\shortparallel\!\!\!\to} v_i \mathrel{\leftarrow\!\!\!\shortparallel\!\!\!-} u_i$ for some v_i. Thus, $t \mathrel{-\!\!\!\shortparallel\!\!\!\to} f(v_1, \ldots, v_n) \mathrel{\leftarrow\!\!\!\shortparallel\!\!\!-} u$ follows.

(iii) If Γ is non-overlapping and P or Q is $\{\epsilon\}$ then the Parallel Moves Lemma applies [1, Lemma 6.4.4]. Figure 1(iii) illustrates the case of $P = \{\epsilon\}$.

(iv) If Γ is overlapping and P or Q is $\{\epsilon\}$, say $P = \{\epsilon\}$, then there exists an overlapping peak $t \overset{\epsilon}{\leftarrow} s \overset{q}{\to} s'$ with $s' \overset{Q'}{\mathrel{-\!\!\!\shortparallel\!\!\!\to}} u$ for $Q' = Q \setminus \{q\}$. By parallel closedness we have $s' \overset{P'}{\mathrel{-\!\!\!\shortparallel\!\!\!\to}} t$ for some P'. As $|t|_{\{\epsilon\}} \geqslant |t|_{P'}$ and $|u|_Q > |u|_{Q'}$ hold, $|t|_{\{\epsilon\}} + |u|_Q > |t|_{P'} + |u|_{Q'}$ holds. Since $(|t|_{\{\epsilon\}} + |u|_Q, s) \succ (|t|_{P'} + |u|_{Q'}, s')$ holds, the induction hypothesis yields $t \mathrel{-\!\!\!\shortparallel\!\!\!\to} \cdot \mathrel{\leftarrow\!\!\!\shortparallel\!\!\!-} u$. ☐

With a small example taken from [2], we illustrate the usage of the theorem.

Example 1. *Consider the left-linear and non-terminating TRS [2]:*

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

While the TRS admits three critical peaks, all of them are closed by single parallel steps:

$$\begin{tikzcd}[column sep=small] & \mathsf{f}(\mathsf{a},\mathsf{a}) \arrow[dl] \arrow[dr] & \\ \mathsf{f}(\mathsf{b},\mathsf{a}) \arrow[rr, dashed, "\shortparallel" description] & & \mathsf{g}(\mathsf{f}(\mathsf{a},\mathsf{a})) \end{tikzcd} \qquad \begin{tikzcd}[column sep=small] & \mathsf{f}(\mathsf{a},\mathsf{a}) \arrow[dl] \arrow[dr] & \\ \mathsf{f}(\mathsf{a},\mathsf{b}) \arrow[rr, dashed, "\shortparallel" description] & & \mathsf{g}(\mathsf{f}(\mathsf{a},\mathsf{a})) \end{tikzcd} \qquad \begin{tikzcd}[column sep=small] & \mathsf{f}(\mathsf{b},\mathsf{b}) \arrow[dl] \arrow[dr] & \\ \mathsf{g}(\mathsf{f}(\mathsf{b},\mathsf{b})) \arrow[rr, dashed, "\shortparallel" description] & & \mathsf{g}(\mathsf{f}(\mathsf{b},\mathsf{b})) \end{tikzcd}$$

Thus, the TRS is parallel closed. Hence, the TRS is confluent.

Using the TRS in Example 1, we compare our induction measure with Huet's original measure [3]. Our proof measures a peak by the amount of *contractums*. Consider e.g. the peak:

$$\mathsf{f}(\overline{\mathsf{b}},\overline{\mathsf{b}}) \overset{\{1,2\}}{\mathrel{\leftarrow\!\!\!\shortparallel\!\!\!-}} \underline{\mathsf{f}(\underline{\mathsf{a}},\underline{\mathsf{a}})} \overset{\{\epsilon\}}{\mathrel{-\!\!\!\shortparallel\!\!\!\to}} \overline{\mathsf{g}(\mathsf{f}(\mathsf{a},\mathsf{a}))}$$

The overlined part indicates the contractums in the target terms of the parallel steps.[1] So the amount is $|\mathsf{b}| + |\mathsf{b}| + |\mathsf{g}(\mathsf{f}(\mathsf{a},\mathsf{a}))| = 6$. Huet's proof [3, 1] measures a peak $t \overset{P_1}{\mathrel{\leftarrow\!\!\!\!\!\!+}} s \overset{P_2}{\mathrel{+\!\!\!\!\!\!\rightarrow}} u$ by the amount of overlaps of *redexes*:

$$|s, P_1, P_2| := \sum_{p \in Q_1 \cup Q_2} \bigl|(s|_p)\bigr|$$

where $Q_1 = \{p \in P_1 \mid p \geqslant q \text{ for some } q \in P_2\}$ and $Q_2 = \{p \in P_2 \mid p > q \text{ for some } q \in P_1\}$. In the above peak the doubly underlined part indicates the overlaps of the redexes. So the amount is 2. Even with this measure the same proof goes through, provided that well-founded induction on $(|s, P, Q|, s)$ with respect to $\succ$ is performed. However, proving the inequality $|s, \{\epsilon\}, Q| > |s', P', Q'|$ in case (iv) is notoriously difficult [5, 4]. This part is trivial in our proof.

3 Toyama's Extensions

Toyama made two variations of Huet's parallel closedness theorem in 1981 [9] and in 1988 [10], but their relation has not been known. In this section we briefly recall his and related results, and then show that Toyama's earlier result subsumes the later one.

In 1988, Toyama showed that the closing form for *overlay* critical pairs, originating from root overlaps, can be relaxed. Let $t \overset{p}{\leftarrow} \cdot \overset{\epsilon}{\rightarrow} u$ be a critical pair. We write $t \overset{\epsilon}{\leftarrow}\!\rtimes\!\rightarrow u$ if $p = \epsilon$, and $t \overset{>\epsilon}{\longleftarrow}\!\rtimes\!\rightarrow u$ if $p > \epsilon$.

Definition 2 ([10])**.** *A TRS is* almost parallel closed *if the inclusions* $\overset{\epsilon}{\leftarrow}\!\rtimes\!\rightarrow \subseteq \mathrel{+\!\!\!\!\!\!\rightarrow} \cdot {}^*\!\!\leftarrow$ *and* $\overset{>\epsilon}{\longleftarrow}\!\rtimes\!\rightarrow \subseteq \mathrel{+\!\!\!\!\!\!\rightarrow}$ *hold.*

Theorem 2 ([10])**.** *A left-linear TRS is confluent if it is almost parallel closed.*

Inspired by almost parallel closedness, Gramlich [2] developed a confluence criterion based on *parallel critical pairs* in 1996.

Definition 3. *We say that* $(\ell\sigma)[r_p]_{p \in P} \mathrel{\leftarrow\!\!\!\!\!\!+} \ell\sigma \overset{\epsilon}{\rightarrow} r\sigma$ *is a* parallel critical peak *of a TRS* $\mathcal{R}$ *if*

- $P \subseteq \mathcal{P}\mathsf{os}_{\mathcal{F}}(\ell)$ *is a non-empty set of parallel positions in* ℓ*,*
- $\ell \rightarrow r$ *and* $\ell_p \rightarrow r_p$ *(for* $p \in P$*) are variants of* $\mathcal{R}$*-rules having no common variables,*
- σ *is a most general unifier of* $\{\ell_p \approx (\ell|_p)\}_{p \in P}$*, and*
- *if* $P = \{\epsilon\}$ *then* $\ell_\epsilon \rightarrow r_\epsilon$ *is not a variant of* $\ell \rightarrow r$*.*

We write $t \overset{>\epsilon}{\mathrel{\leftarrow\!\!\!\!\!\!+}}\!\rtimes\!\rightarrow u$ *if* $t \overset{P}{\mathrel{\leftarrow\!\!\!\!\!\!+}} s \overset{\epsilon}{\rightarrow} u$ *is a parallel critical peak and* $P \neq \{\epsilon\}$*.*

Theorem 3 ([2])**.** *A left-linear TRS is confluent if the inclusions* $\leftarrow\!\rtimes\!\overset{\epsilon}{\rightarrow} \subseteq \mathrel{+\!\!\!\!\!\!\rightarrow} \cdot {}^*\!\!\leftarrow$ *and* $\overset{>\epsilon}{\mathrel{\leftarrow\!\!\!\!\!\!+}}\!\rtimes\!\rightarrow \subseteq \rightarrow^*$ *hold.*

Unfortunately, this criterion by Gramlich does not subsume (almost) parallel closedness.

Example 2 (Continued from Example 1)**.** *The TRS admits the following non-overlay parallel critical peak* $\mathsf{f}(\mathsf{b},\mathsf{b}) \mathrel{\leftarrow\!\!\!\!\!\!+} \mathsf{f}(\mathsf{a},\mathsf{a}) \overset{\epsilon}{\rightarrow} \mathsf{g}(\mathsf{f}(\mathsf{a},\mathsf{a}))$*. However,* $\mathsf{f}(\mathsf{b},\mathsf{b}) \rightarrow^* \mathsf{g}(\mathsf{f}(\mathsf{a},\mathsf{a}))$ *does not hold.*

[1] Up to our best knowledge, this idea first appeared in the paper by Oyamaguchi and Ohta [7].

As noted in the paper [2], Toyama [9] had already obtained in 1981 a closedness result that subsumes Theorem 3.

Theorem 4 ([9])**.** *A left-linear TRS is confluent if the following conditions hold.*

(a) The inclusion $\leftarrow\!\rtimes\!\xrightarrow{\epsilon} \subseteq \mathrel{\shortparallel\!\!\!\!\!\rightarrow} \cdot {}^*\!\!\leftarrow$ *holds.*

(b) Every parallel critical peak $t \overset{P}{\mathrel{\leftarrow\!\!\!\!\!\shortparallel}} s \xrightarrow{\epsilon} u$ *admits a valley of the form* $t \to^* v \overset{Q}{\mathrel{\leftarrow\!\!\!\!\!\shortparallel}} u$ *with* $\mathcal{V}\mathsf{ar}(s,P) \supseteq \mathcal{V}\mathsf{ar}(v,Q)$*. Here* $\mathcal{V}\mathsf{ar}(s,P)$ *stands for* $\bigcup_{p \in P} \mathcal{V}\mathsf{ar}(s|_p)$*.*

Example 3. *Consider again the TRS of Example 1. As witnessed in Example 1, the inclusion* $\leftarrow\!\rtimes\!\xrightarrow{\epsilon} \subseteq \mathrel{\shortparallel\!\!\!\!\!\rightarrow}$ *holds. Thus, condition (a) of Theorem 4 holds. There are four parallel critical peaks and they can be closed as follows:*

$$\begin{tikzcd}
 & \mathsf{f(a,a)} \arrow[dl, "\shortparallel" marking] \arrow[dr] & \\
\mathsf{f(b,a)} \arrow[rr, dashed, bend right, "*"'] & & \mathsf{g(f(a,a))}
\end{tikzcd}
\quad
\begin{tikzcd}
 & \mathsf{f(a,a)} \arrow[dl, "\shortparallel" marking] \arrow[dr] & \\
\mathsf{f(a,b)} \arrow[rr, dashed, bend right, "*"'] & & \mathsf{g(f(a,a))}
\end{tikzcd}
\quad
\begin{tikzcd}
 & \mathsf{f(b,b)} \arrow[dl, "\shortparallel" marking] \arrow[dr] & \\
\mathsf{g(f(b,b))} \arrow[rr, dashed, bend right, "*"'] & & \mathsf{g(f(b,b))}
\end{tikzcd}
\quad
\begin{tikzcd}
 & \mathsf{f(a,a)} \arrow[dl, "\shortparallel" marking] \arrow[dr] & \\
\mathsf{f(b,b)} \arrow[dr, dashed, "*"'] & & \mathsf{g(f(a,a))} \arrow[dl, dashed, "\shortparallel" marking] \\
 & \mathsf{g(f(b,b))} &
\end{tikzcd}$$

It is easy to see that the diagrams fulfil condition (b). Hence, confluence follows by Theorem 4.

We show that Theorem 4 even subsumes Theorem 2. The next lemma relates the Parallel Moves Lemma [1, Lemma 6.4.4] to the variable condition of Theorem 4.

Lemma 1. *Let* $\mathcal{R}$ *be a left-linear TRS. If* $t \overset{P}{\mathrel{\leftarrow\!\!\!\!\!\shortparallel}} s \xrightarrow{\epsilon} u$ *is non-overlapping then* $t \xrightarrow{\epsilon} v \overset{Q}{\mathrel{\leftarrow\!\!\!\!\!\shortparallel}} u$ *and* $\mathcal{V}\mathsf{ar}(s,P) \supseteq \mathcal{V}\mathsf{ar}(v,Q)$ *for some term* v *and set* Q *of parallel positions.*

The above statement is extended to parallel peaks for almost parallel closed TRSs.

Lemma 2. *Let* $\mathcal{R}$ *be a left-linear almost parallel closed TRS. If* $t \overset{P_1}{\mathrel{\leftarrow\!\!\!\!\!\shortparallel}} s \overset{P_2}{\mathrel{\shortparallel\!\!\!\!\!\rightarrow}} u$ *then*

- $t \to^* v_1 \overset{Q_1}{\mathrel{\leftarrow\!\!\!\!\!\shortparallel}} u$ *and* $\mathcal{V}\mathsf{ar}(s,P_1) \supseteq \mathcal{V}\mathsf{ar}(v_1,Q_1)$ *for some* v_1 *and* Q_1*, and*
- $t \overset{Q_2}{\mathrel{\shortparallel\!\!\!\!\!\rightarrow}} v_2 \; {}^*\!\!\leftarrow u$ *and* $\mathcal{V}\mathsf{ar}(s,P_2) \supseteq \mathcal{V}\mathsf{ar}(v_2,Q_2)$ *for some* v_2 *and* Q_2*.*

Proof. Let $\Gamma\colon t \overset{P_1}{\mathrel{\leftarrow\!\!\!\!\!\shortparallel}} s \overset{P_2}{\mathrel{\shortparallel\!\!\!\!\!\rightarrow}} u$. We perform well-founded induction on $(|t|_{P_1} + |u|_{P_2}, s)$ with respect to $\succ$. We distinguish cases, depending on the shape of Γ.

(i) If P_1 or P_2 is $\varnothing$ then the claim follows from the fact: $\mathcal{V}\mathsf{ar}(w,P) \supseteq \mathcal{V}\mathsf{ar}(v,P)$ if $w \overset{P}{\mathrel{\shortparallel\!\!\!\!\!\rightarrow}} v$.

(ii) If $P_1 \nsubseteq \{\epsilon\}$ and $P_2 \nsubseteq \{\epsilon\}$ then we may assume $s = f(s_1,\dots,s_n)$, $t = f(t_1,\dots,t_n)$, $u = f(u_1,\dots,u_n)$, and $t_i \overset{P_1^i}{\mathrel{\leftarrow\!\!\!\!\!\shortparallel}} s_i \overset{P_2^i}{\mathrel{\shortparallel\!\!\!\!\!\rightarrow}} u_i$ for all $1 \leqslant i \leqslant n$. Here P_k^i denotes the set $\{p \mid i{\cdot}p \in P_k\}$. As in case (ii) of the proof of Theorem 1 we can deduce $(|t|_{P_1} + |u|_{P_2}, s) \succ (|t_i|_{P_1^i} + |u_i|_{P_2^i}, s_i)$. Consider an i-th peak $t_i \overset{P_1^i}{\mathrel{\leftarrow\!\!\!\!\!\shortparallel}} s_i \overset{P_2^i}{\mathrel{\shortparallel\!\!\!\!\!\rightarrow}} u_i$. By the induction hypothesis it admits valleys of the forms $t_i \overset{Q_1^i}{\mathrel{\shortparallel\!\!\!\!\!\rightarrow}} v_1^i \; {}^*\!\!\leftarrow u_i$ and $t_i \to^* v_2^i \overset{Q_2^i}{\mathrel{\leftarrow\!\!\!\!\!\shortparallel}} u_i$ such that $\mathcal{V}\mathsf{ar}(v_k^i, Q_k^i) \subseteq \mathcal{V}\mathsf{ar}(s_i, P_k^i)$ for both $k \in \{1,2\}$. For each k, take $Q_k = \{i \cdot q \mid 1 \leqslant i \leqslant n \text{ and } q \in Q_k^i\}$ and $v_k = f(v_k^1,\dots,v_k^n)$. Then, $t \to^* v_1 \overset{Q_1}{\mathrel{\leftarrow\!\!\!\!\!\shortparallel}} u$ and $t \overset{Q_2}{\mathrel{\shortparallel\!\!\!\!\!\rightarrow}} v_2 \; {}^*\!\!\leftarrow u$ hold, and moreover the inclusion

$$\mathcal{V}\mathsf{ar}(v_k, Q_k) = \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(v_k^i, Q_k^i) \subseteq \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(s_i, P_k^i) = \mathcal{V}\mathsf{ar}(s, P_k)$$

holds for each k. Hence, the claim follows.

(iii) If Γ is non-overlapping and P_1 or P_2 is $\{\epsilon\}$ then the claim is straightforward from Lemma 1.

(iv) If Γ is overlapping with $P_1 = P_2 = \{\epsilon\}$ then by almost parallel closedness $t \to^* v_1 \overset{Q_1}{\leftarrow\!\!\!\!\!\!\parallel} u$ and $t \overset{Q_2}{\parallel\!\!\!\!\!\!\rightarrow} v_2 \; {}^*\!\!\leftarrow u$ for some v_1, v_2, Q_1, and Q_2. For each $k \in \{1, 2\}$ we have $s \to^* v_k$, so $\mathcal{V}\mathsf{ar}(v_k) \subseteq \mathcal{V}\mathsf{ar}(s)$ follows. Therefore, $\mathcal{V}\mathsf{ar}(v_k, Q_k) \subseteq \mathcal{V}\mathsf{ar}(v_k) \subseteq \mathcal{V}\mathsf{ar}(s) = \mathcal{V}\mathsf{ar}(s, \{\epsilon\})$. The claim holds.

(v) If Γ is overlapping with $P_1 = \{\epsilon\}$ and $P_2 \neq \{\epsilon\}$ then there exists $p \in P_2$ such that $s \xrightarrow{p} s' \overset{P_2 \setminus \{p\}}{\parallel\!\!\!\!\!\!\rightarrow} u$ and $t \xleftarrow{\epsilon} s \xrightarrow{p} s'$ is an instance of a critical peak. By almost parallel closedness $t \overset{P_1'}{\leftarrow\!\!\!\!\!\!\parallel} s'$ for some P_1'. As in case (iv) of the proof of Theorem 1 we can deduce $(|t|_{P_1} + |u|_{P_2}, s) \succ (|t|_{P_1'} + |u|_{P_2 \setminus \{\epsilon\}}, s')$. Thus, the claim follows by the induction hypothesis for $t \overset{P_1'}{\leftarrow\!\!\!\!\!\!\parallel} s' \overset{P_2 \setminus \{p\}}{\parallel\!\!\!\!\!\!\rightarrow} u$ and $\mathcal{V}\mathsf{ar}(s', P_2 \setminus \{p\}) \subseteq \mathcal{V}\mathsf{ar}(s, P_2)$. $\square$

Theorem 5. *Every left-linear and almost parallel closed TRS satisfies conditions (a) and (b) of Theorem 4. In other words, Theorem 4 subsumes Theorem 2.*

Proof. Since (parallel) critical peaks are instances of $\leftarrow\!\!\!\!\!\!\parallel \cdot \parallel\!\!\!\!\!\!\rightarrow$, Lemma 2 entails the claim. $\square$

4 Concluding Remark

We presented a simple proof of Huet's parallel closedness theorem, using a new measure, and also proved that Toyama's almost parallel closedness theorem is subsumed by his earlier result based on parallel critical pairs. We anticipate that our measure can be adapted to other confluence criteria for term rewriting and conditional rewriting, while it is unclear whether it can be adapted to the proof of van Oostrom's development closedness [6]. At least the current definition does not go through. Despite its powerfulness, Theorem 4 has not been well studied. In particular, whether the variable condition of Theorem 4 is essential is our primary question.

References

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

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

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

[4] J. Nagele. *Mechanizing Confluence.* PhD thesis, University of Innsbruck, 2017.

[5] J. Nagele and A. Middeldorp. Certification of classical confluence results for left-linear term rewrite systems. In *Proc. 7th ITP*, volume 9807 of *LNCS*, pages 290–306, 2016.

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

[7] M. Oyamaguchi and Y. Ohta. A new parallel closed condition for Church-Rosser of left-linear term rewriting systems. In *Proc. 10th RTA*, volume 1232 of *LNCS*, pages 187–201, 1997.

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

[9] Y. Toyama. On the Church-Rosser property of term rewriting systems. Technical Report 17672, NTT Electro-Technical Laboratory, 1981. In Japanese.

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

Algebraic critical pair lemma

Cyrille Chenavier, Benjamin Dupont, and Philippe Malbos

[1] Johanes Kepler University, Linz, Austria, cyrille.chenavier@jku.at
[2,3] Université de Lyon, France, {bdupont,malbos}@math.univ-lyon1.fr

Abstract

Convergent rewriting systems on algebraic structures give methods to prove coherence results and compute homological invariants of these structures. These methods are based on higher-dimensional extensions of the critical pair lemma that characterizes local confluence from confluence of critical pairs. The analysis of local confluence of rewriting systems on algebraic structures, such as groups or linear algebras, is complicated because of the underlying algebraic axioms, and local confluence properties require additional termination conditions. In this work, we define the structure of algebraic polygraph modulo that formalizes the interaction between the rules of the rewriting system and the inherent algebraic axioms, and we show a critical pair lemma algebraic polygraphs. We deduce from this result a critical pair lemma for rewriting systems on algebraic structures specified by rewriting systems convergent modulo AC. As an illustration, we explicit our constructions on linear rewriting systems.

1 Introduction

The critical-pair completion (CPC) is an approach developed in the mid sixties that combines completion procedure and the notion of critical pair [2]. It originates from theorem proving [12], polynomial ideal theory [1], and the word problem [9, 11]. In the mid eighties, CPC has found original and deep applications in algebra to solve coherence problems [14], or to compute homological invariants [13]. More recently, higher-dimensional extensions of the CPC approach were used for the computation of cofibrant replacements of algebraic and categorical structures [5, 6]. These constructions based on CPC are known for monoids, small categories, and algebras over a field. However, the extension of these methods to a wide range of algebraic structures is made difficult because of the interaction between the rewriting rules and the inherent axioms of the algebraic structure. For this reason, the higher-dimensional extensions of the CPC approach for a wide range of algebraic structures, including groups, Lie algebras, is still an open problem.

One of the main tools to reach confluence in CPC procedure for algebraic rewriting systems is the critical pair lemma, or critical branching lemma (CBL). Its proof is based on classification of the local branchings into *orthogonal* branchings, that is involving two rules that do not overlap, *overlapping* branchings involving two rules that overlap. A *critical branching* is a minimal overlapping application of two rules on the same redex. When the orthogonal branchings are confluent, if all critical branchings are confluent, then local confluence holds. Thus, the main argument to achieve CBL is to prove that orthogonal and overlapping branchings are confluent. For string and term rewriting systems, orthogonal branchings are always confluent, and confluence of critical branchings implies confluence of overlapping branchings. The situation is more complicated for rewriting systems on a linear structure.

The well known approaches of rewriting in the linear context consist in orienting the rules with respect to an ambient monomial order, and CBL is well known in this context. However, some algebras do not admit any higher-dimensional finite convergent presentation on a fixed set of generators with respect to a monomial order, [5]. However, when the orientation of rules does not depend on a monomial order, as in [5], the CBL requires additional assumptions, namely termination and positivity of reductions. A positive reduction for a linear rewriting system, as defined in [5], is the application of a reduction rule on a monomial that does not appear in the polynomial context. For instance, consider the linear rewriting system on an associative algebra over a field $\mathbb{K}$ given in [5] defined by rules $\alpha : xy \to xz$ and $\beta : zt \to 2yt$. It has no critical branching, but it has a non-confluent additive branching:

$$\begin{tikzcd}
& & 4xyt \arrow[r, "4\alpha t"] & 4xzt \arrow[r, "4x\beta"] & \cdots \\
& 2xzt \arrow[ur, bend left, "2x\beta"] \arrow[dr, dotted, "xzt + x\beta"] & & & \\
xyt + xzt \arrow[ur, bend left, "\alpha t + xzt"] \arrow[dr, bend right, "xyt + x\beta"'] & = & xzt + 2xyt & & \\
& 3xyt \arrow[ur, dotted, "\alpha t + 2xyt"'] \arrow[dr, bend right, "3\alpha t"'] & & & \\
& & 3xzt \arrow[r, "3x\beta"'] & 6xyt \arrow[r, "6\alpha t"'] & \cdots
\end{tikzcd}$$

The dotted arrows correspond to non positive reductions. We note that the lack of termination is an obstruction to confluence of orthogonal branchings.

In this work we introduce an algebraic setting for the formulation of the CBL. We define the structure of algebraic polygraph modulo which formalizes the interaction between the rules of the rewriting system and the inherent axioms of the algebraic structure. We show a CBL for algebraic polygraphs modulo. We deduce from this result a CBL for rewriting systems on algebraic structures whose axioms are specified by term rewriting systems that are convergent modulo associativity and commutativity. Finally, we explicit our results in linear rewriting, and explain why termination is a necessary condition to characterize local confluence in that case.

In Section 2, we recall the categorical structure of cartesian polygraph introduced in [10]. In Section 3, we introduce the notion of algebraic polygraph modulo, and we refer the reader to [4] for a categorical interpretation of the given constructions. In Section 4, we present confluence property of algebraic polygraphs modulo from [4] and algebraic polygraphs modulo with respect a positive strategy σ. Finally, we state the algebraic critical branching lemma. This abstract is a short version of the preprint [3], where more detailed constructions and examples can be found.

2 Cartesian polygraphs

A *signature* is defined by a set P_0 of *sorts* and a set P_1 of *operations* on the free monoid over P_0. We denote by $s_0(\alpha)$ and $t_0(\alpha)$ the *arity* and *coarity* of $\alpha \in P_1$. When $s_1, \ldots, s_k$ are sorts, we denote $\underline{s} = s_1 \ldots s_k$ their product in the free monoid over P_0. We denote by $P_1^{\times}$ the free theory generated by a signature (P_0, P_1). Its 1-cells, also called *terms* on the signature (P_0, P_1) are defined inductively. The canonical projections $x_i^{\underline{s}} : \underline{s} \to s_i$ are *variables*, and for any terms $f : \underline{s} \to r$ and $f' : \underline{s} \to r'$ in $P_1^{\times}$, we denote by $\langle f, f' \rangle : \underline{s} \to rr'$, the *pairing* of terms f, f'. A *(cartesian) 2-polygraph* is a data made of a signature (P_0, P_1), and a set P_2 equipped with two maps $t_1, s_1 : P_2 \to P_1^{\times}$, satisfying the *globular conditions* $s_0 s_1 = s_0 t_1$ and $t_0 s_1 = t_0 t_1$. An element α of P_2 is called a *rule*, and relates terms of same arity and coarity.

2.1. Two-dimensional theories. Recall that a *2-theory* is a 2-category with an additional cartesian structure on its 1-cells and 2-cells [10]. We denote by $P_2^{\times}$ the free 2-theory generated by a cartesian 2-polygraph (P_0, P_1, P_2). Its underlying 1-category is the free theory $P_1^{\times}$, and its 2-cells are defined inductively as follows. For $\alpha : f \Rightarrow f'$ in P_2 and $h \in P_1^{\times}$, there is a 2-cell $\alpha h : f \star h \Rightarrow f' \star h$ in $P_2^{\times}$, and for $\beta : g \Rightarrow g'$ in $P_2^{\times}$, there is a 2-cell $\langle \alpha, \beta \rangle : \langle f, g \rangle \Rightarrow \langle f', g' \rangle$.

Finally, there are 2-cells in $P_2^\times$ of the form $A[\alpha] : A[f] \Rightarrow A[f']$ where $A[\Box]$ denotes an *algebraic context* of the form: $k\langle \mathrm{id}_{k_1}, \ldots, \Box_i, \ldots, \mathrm{id}_{k_j}\rangle : \underline{s} \to r$, where $k_1, \ldots, k_j : \underline{s} \to \underline{r_i}$ and $k : \underline{r} \to r$ belong to $P_1^\times$, and $\Box_i$ is the i-th element of the pairing. These 2-cells are required to satisfy *exchange relations*, see [10]. The source and target maps s_1, t_1 extend to $P_2^\times$ and we denote a_- and a_+ for $s_1(a)$ and $t_1(a)$. A *ground term* in $P_1^\times$ is a term with source $\mathbf{0}$. A 2-cell a in $P_2^\times$ is *ground* when a_- is a ground term. An algebraic context $A[\Box] = f\langle f_1, \ldots, \Box_i, \ldots f_{|\underline{r}|}\rangle$ is called *ground* when the f_i are ground terms.

The free $(2,1)$-theory generated by a cartesian 2-polygraph (P_0, P_1, P_2), denoted by $P_2^\top$, is the free 2-theory $P_2^\times$ whose any 2-cell is invertible with respect the $\star$-composition. The 2-cells of the $(2,1)$-theory $P_2^\top$ corresponds to elements of the equivalence relation generated by P_2.

2.2. Rewriting properties of cartesian polygraphs. The algebraic contexts of a 2-polygraph P can be composed as follows $AA'[\Box] := A[A'[\Box]]$. One defines a *bi-context* as $B[\Box_i, \Box_j] := f\langle \mathrm{id}_{f_1}, \ldots, \Box_i, \ldots, \Box_j, \ldots, \mathrm{id}_{f_k}\rangle$ where the $f_k : \underline{s} \to \underline{r_k}$ and $f : \underline{r} \to r$ are terms in $P_1^\times$, and $\Box_i$ (resp. $\Box_j$) has to be filled by a term $g_i : \underline{s} \to \underline{r_i}$ (resp. $g_j : \underline{s} \to \underline{r_j}$). A 2-cell of the form $A[\alpha w]$ where A is an algebraic context, w is a term in $P_1^\times$ and $\alpha \in P_2$ is called a P-*rewriting step*. A P-*rewriting path* is a non-identity 2-cell of $P_2^\times$. Such a 2-cell can be decomposed as a $\star$-composition of rewriting steps $\alpha = A_1[\alpha_1 w_1] \star \ldots A_k[\alpha_k w_k]$.

3 Algebraic polygraphs modulo

Let (P_0, P_1) be a signature, and Q be a set of *generating ground terms* whose target is a sort in P_0. We denote by $P_1\langle Q\rangle$ the set of ground terms of the free theory $(P_1 \cup Q)^\times$. An *algebraic polygraph* is a data made of a 2-polygraph P, a family of set of constant 1-cells Q, and a cellular extension R of the set of ground terms $P_1\langle Q\rangle$, that is a set equipped with two source and target maps $R \to P_1\langle Q\rangle$. A R-*rewriting step* is a ground 2-cell in the free 2-theory $R^\times$ of the form $A[\alpha] : A[f] \to A[g]$, where $A[\Box]$ is a ground context. A R-*rewriting path* is a finite or infinite sequence $a_1 \star \ldots \star a_k \star \ldots$ of R-rewriting steps a_i. The *size* of a R-rewriting path $\underline{a}$, denoted by $|\underline{a}|$, is the number of rewriting steps needed to write $\underline{a}$ as a composition as above. The cellular extension P_2 defined on $P_1^\times$ extends to a cellular extension on the free 1-theory $(P_1 \cup Q)^\times$ denoted by $\widehat{P}_2$. We denote by $P_2\langle Q\rangle$ the set of ground 2-cells in the free 2-theory $(\widehat{P}_2)^\times$. The algebraic polygraph $(P, Q, P_2\langle Q\rangle)$ is called the *algebraic polygraph of axioms*. We denote by $\overline{P\langle Q\rangle}$ the quotient of $P_1\langle Q\rangle$ by the congruence generated by relations in $P_2\langle Q\rangle$.

3.1. Positive strategies. Denote by $\overline{f}$ the image of a ground term f by the canonical projection $\pi : P_1\langle Q\rangle \to \overline{P\langle Q\rangle}$. Let $\sigma : \overline{P\langle Q\rangle} \to \mathbf{Set}$ be a map such that for any $\overline{f} \in \overline{P\langle Q\rangle}$, $\sigma(\overline{f})$ is a chosen non-empty subset of $\pi^{-1}(\overline{f})$. Such a map is called a *positive strategy* with respect to (P, Q). A R-rewriting step a is called σ-*positive* if a_- belongs to $\sigma(\overline{a_-})$, and a R-rewriting path $a_1 \star \ldots \star a_k$ is called σ-*positive* if any of its rewriting steps is positive.

We will use positive strategies wrt a 2-polygraph P such that $P_2 = P_2' \cup P_2''$, with P_2' confluent modulo P_2''. For every 1-cell $\overline{f}$ in $\overline{P\langle Q\rangle}$, we set $\sigma(\overline{f}) = \mathrm{NF}(f, P_2' \bmod P_2'')$, where $f \in \pi^{-1}(\overline{f})$, the set of normal forms of f for P_2' modulo P_2''. This is well-defined following [7], since if $f, f' \in \pi^{-1}(\overline{f})$, then $\mathrm{NF}(f, P_2' \bmod P"_2) \equiv_{P"_2} \mathrm{NF}(f', P_2' \bmod P"_2)$.

3.2. Algebraic polygraphs modulo. Given an algebraic polygraph $\mathcal{P} = (P, Q, R)$ and a positive strategy σ on $\mathcal{P}$, one denotes by ${}_P R_P$ the cellular extension of $P_1\langle Q\rangle$ whose elements are of the form $e \star a \star e'$, where e and e' are 2-cells in $P_2\langle Q\rangle^\top$ and a is a R-rewriting step such that $e_+ = a_-$ and $a_+ = e'_-$, see [4] for a detailed construction. A 2-cell $e \star a \star e'$ in ${}_PR_P$ is called σ-positive if a is a σ-positive R-rewriting step. An *algebraic polygraph modulo*, APM for short, is a data (P, Q, R, S) made of an algebraic polygraph (P, Q, R), and a cellular extension S of $P_1\langle Q\rangle$ such that $R \subseteq S \subseteq {}_PR_P$.

An algebraic polygraph (P, Q, R) is called *quasi-terminating* if for each sequence $(f_n)_{\in\mathbb{N}}$ of 1-cells of $P_1\langle Q\rangle$ such that for each $n \in \mathbb{N}$, there is a rewriting step $f_n \to f_{n+1}$, the sequence $(f_n)_{\in\mathbb{N}}$ contains an infinite number of occurrences of same 1-cell. An APM (P, Q, R, S) is called *quasi-terminating* if the algebraic polygraph (P, Q, S) is quasi-terminating. A 1-cell f of $P_1\langle Q\rangle$ is *quasi-irreducible* if for any S-rewriting step $f \to g$, there exists a S-rewriting sequence from g to f. A *quasi-normal form* of a 1-cell f in $P_1\langle Q\rangle$ is a quasi-irreducible 1-cell $\widetilde{f}$ of $P_1\langle Q\rangle$ such that there exists a S-rewriting sequence from f to $\widetilde{f}$. For a quasi-terminating APM, any 1-cell f of $P_1\langle Q\rangle$ admits at least a quasi-normal. A *quasi-normal form strategy* is a map $s : P_1\langle Q\rangle \to P_1\langle Q\rangle$ sending a 1-cell f on a chosen quasi-normal $\widetilde{f}$.

An *algebraic rewriting system* on (P, Q, R, S, σ) is a cellular extension $\overline{S}$ of $\overline{P}\langle Q\rangle$ defined by $\overline{S} = \{\overline{a} : \overline{a_-} \Rightarrow \overline{a_+} \mid a \in S\}$. Let us consider the subset $\overline{S}^{\sigma}$ of $\overline{S}$ defined by $\overline{S}^{\sigma} = \{\overline{a} : \overline{a_-} \Rightarrow \overline{a_+} \mid a$ is a σ-positive S-rule$\}$. A $\overline{S}$*-rewriting step* (resp. a $\overline{S}^{\sigma}$-rewriting step) is a 2-cell in $\overline{S}^{\times}$ (resp. $\overline{S}^{\sigma\times}$) of the form $\overline{C}[\overline{a}] : \overline{C}[\overline{a_-}] \Rightarrow \overline{C}[\overline{a_+}]$, where C is a ground context of $P_1\langle Q\rangle$ and $C[a]$ is a S-rewriting step (resp. σ-positive S-rewriting step). A $\overline{S}$*-rewriting path* is a sequence of $\overline{S}$-rewriting steps.

4 Confluence in algebraic polygraphs modulo

Let $\mathcal{P} = (P, Q, R, S)$ be an APM with a positivity strategy σ. A σ-*branching* of $\mathcal{P}$ is a triple (a, e, b) where a, b are σ-positive 2-cells of $S^{\times}$ and e is a 1-cell of $P_2\langle Q\rangle^{\top}$ such that $e_- = a_-$ and $e_+ = b_-$. It is *local* if a is a S-rewriting step, b is a 2-cell of $S^{\times}$ and e a 2-cell of $P_2\langle Q\rangle^{\top}$ such that $|e| + |b| = 1$. Note that the 2-cell b (resp. e) can be an identity 2-cell of $S^{\times}$ (resp of $P_2\langle Q\rangle^{\top}$), and in that case the σ-branching is of the form (a, e) (resp. (a, b)). Such a σ-branching is σ-*confluent modulo* if there exist σ-positive 2-cells a' and b' in $S^{\times}$ and a 2-cell e' of $P_2\langle Q\rangle^{\top}$ as depicted on the right. We say that the APM $\mathcal{P}$ is *confluent modulo* (resp. *locally confluent modulo*) if any σ-branching modulo (resp. local branching modulo) is confluent modulo.

$$\begin{tikzcd}
f \arrow[r, "a"] \arrow[d, "e"'] & f' \arrow[r, dotted, "a'"] & h \arrow[d, "e'"] \\
g \arrow[r, "b"'] & g' \arrow[r, dotted, "b'"'] & h'
\end{tikzcd}$$

4.1. Theorem (Newman lemma modulo for algebraic polygraphs modulo). *Let $\mathcal{P}$ be a quasi-terminating APM, and σ be a positive strategy on $\mathcal{P}$. If $\mathcal{P}$ is locally σ-confluent modulo, then it is σ-confluent modulo.*

The proof of this result, and of Thm. 4.4 are based on the principle of double induction on the distance to the quasi-normal form, and are extensions to quasi-terminating setting of Huet's constructions based on double induction principle [7] in the terminating setting. Let $d : P_1\langle Q\rangle \to \mathbb{N}$ maping a 1-cell f to the length $d(f)$ of the shortest ${}_PR_P$-rewriting path from f to $\widetilde{f}$, that we extend to a map on σ-branchings (a, e, b) by setting $d(a, e, b) := d(a_-) + d(b_-)$. We define a well-founded order $\prec$ on the set of σ-branchings of $\mathcal{P}$ by $(a, e, b) \prec (a', e', b')$ if $d(a, e, b) < d(a', e', b')$.

4.2. Classification of local σ-branchings modulo. The local σ-branchings modulo of S can be classified in the following families:

$$A[a_+] \xleftarrow{a} A[a_-] \xrightarrow{a} A[a_+] \qquad\qquad A[a_+] \xleftarrow{a} A[a_-] = A[A'[b_-]] \xrightarrow{b} A[A'[b_+]]$$

Trivial *Inclusion independant*

$$\begin{tikzcd} B[a_-, b_-] \arrow[r, "B[a,b_-]"] \arrow[d, "||"'] & B[a_+, b_-] \\ B[a_-, b_-] \arrow[r, "B[a_-,b]"'] & B[a_-, b_+] \end{tikzcd} \qquad \begin{tikzcd} B[a_-, e_-] \arrow[r, "B[a,e_-]"] \arrow[d, "B[a_-,e]"'] & B[a_+, e_-] \\ B[a_-, t_1(e)] & \end{tikzcd}$$

Orthogonal *Orthogonal modulo*

together with symmetries on orthogonal σ-branchings modulo, for some σ-positive S-rewriting steps a, b, 2-cell e in $P_2\langle Q\rangle^\top$, ground contexts A, A', and ground bi-contexts B, B'. The remaining local σ-branchings modulo are called *non-orthogonal* σ-branchings modulo.

We define an order relation on σ-branchings modulo of $\mathcal{P} = (P, Q, R, S)$ by setting $(a, e, b) \sqsubseteq (A[a], A[e], A[b])$ for a ground context A. A *critical σ-branching modulo* is a local σ-branching modulo P which is non trivial, non orthogonal and minimal wrt the order relation $\sqsubseteq$.

4.3. Positive confluence. An APM (P, Q, R, S) with a positive strategy σ is called *σ-positively confluent* if, for any S-rewriting step f, there exists a representing $\widetilde{a_-} \in \sigma(a_-)$ of a_- and two σ-positive S-reductions a' and b' of size at most 1 as in the following on the right.

$$\begin{tikzcd} \widetilde{a_-} \arrow[rrr, "a'"] \arrow[d, "e"'] & & & {} \arrow[d, "e''"] \\ a_- \arrow[r, "a"'] & {} \arrow[r, "e'"'] & {} \arrow[r, "b'"'] & {} \end{tikzcd}$$

4.4. Theorem (Terminating critical branching theorem modulo). *Let (P, Q, R, S) be a quasi-terminating and positively σ-confluent APM with a positive strategy σ. Then it is locally σ-confluent modulo if and only if the two following properties hold:*

$\mathbf{a_0}$) *any critical σ-branching modulo (a, b) made of S-rewriting steps is σ-confluent modulo.*

$\mathbf{b_0}$) *any critical σ-branching modulo (a, e), with a S-rewriting step and e is a 2-cell in $P_2\langle Q\rangle^\top$ of length 1, is σ-confluent modulo.*

When all the reductions are positive, that is $S(\overline{u}) = \pi^{-1}(\overline{u})$ for any $\overline{u}$, the quasi-termination assumption in Prop. 4.4 are not needed. In that case, the positive confluence is always satisfied.

4.5. Algebraic critical branching lemma. Let $\mathcal{A}$ be an algebraic rewriting system on an APM $\mathcal{P} = (P, Q, R, S)$. The *critical branchings* of $\mathcal{A}$ are the projections of the critical σ-branchings modulo of $\mathcal{P}$ of the form $\mathbf{a}_0$), that is pairs $(\overline{a}, \overline{b})$ of $\overline{S}^\sigma$-rewriting steps such that there is a σ-branching modulo in $\mathcal{P}$ with source $(\widetilde{a_-}, \widetilde{b_-})$. From Prop. 4.4, we deduce our main result.

4.6. Theorem. *Let $\mathcal{P} = (P, Q, R, S)$ be an APM such that ${}_P R_P$ is quasi-terminating and positively confluent. Let $\mathcal{A}$ be an algebraic rewriting system on $\mathcal{P}$. Then $\mathcal{A}$ is locally confluent if and only if its critical branchings are confluent.*

4.7. CBL for linear rewriting. Suppose that P contains the convergent 2-polygraph modulo AC that presents the theory of modules over commutative rings given in [8], denoted by RMod. If P_2'' is the 2-polygraph of associativity and commutativity relations, and P_2' is RMod, then Thm. 4.6 corresponds to CBL for linear rewriting systems proved in [5]. Indeed, given an APM (P, Q, R, S) with the σ-strategy defined in 3.1, one proves that the positivity confluence of S with respect to σ implies the factorization property of [5]. This property means that any rewriting step a of $\overline{S}$ can be decomposed as $a = b \star c^{-1}$ where b and c are either rewriting steps of $\overline{S}^\sigma$ or identities. Finally, the quasi-termination assumption of ${}_P R_P$ is equivalent to the termination assumption in [5].

References

[1] B. Buchberger. *An Algorithm for Finding the Basis Elements in the Residue Class Ring Modulo a Zero Dimensional Polynomial Ideal.* PhD thesis, University of Innsbruck, Austria, 1965.

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

[3] Cyrille Chenavier, Benjamin Dupont, and Philippe Malbos. Algebraic polygraphs modulo and linear rewriting. preprint, arXiv:2004.14361, 2020.

[4] B. Dupont and P. Malbos. Coherent confluence modulo relations and double groupoids. arXiv:1810.08184, Hal-01898868, 2018.

[5] Y. Guiraud, E. Hoffbeck, and P. Malbos. Convergent presentations and polygraphic resolutions of associative algebras. *Math. Z.*, 2019.

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

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

[8] J.-M. Hullot. A catalogue of canonical term rewriting systems. 1980. SRI International, Technical Report CSL 113.

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

[10] P. Malbos and S. Mimram. Cartesian polygraphic resolutions. in progress, 2020.

[11] M. Nivat. Congruences parfaites et quasi-parfaites. In *Séminaire P. Dubreil, 25e année (1971/72), Algèbre, Fasc. 1, Exp. No. 7*, page 9. Secrétariat Mathématique, Paris, 1973.

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

[13] C. Squier and F. Otto. The word problem for finitely presented monoids and finite canonical rewriting systems. In *RTA, 1987*, volume 256 of *LNCS*, pages 74–82. 1987.

[14] C. C. Squier, F. Otto, and Y. Kobayashi. A finiteness condition for rewriting systems. *TCS*, 131(2):271–294, 1994.

On the reduction of the type-free computational λ-calculus

Ugo de'Liguoro and Riccardo Treglia

Università di Torino, Turin, Italy
ugo.deliguoro@unito.it
riccardo.treglia@unito.it

Abstract

We study the reduction of the computational λ-calculus in the untyped case. To this aim, we consider a minimal fragment of the λ-calculus with monads as introduced by Wadler, and define a notion of call-by-value reduction just by orienting the three monad equational laws. We then prove confluence of its compatible closure. Finally, we show factorization of any reduction sequence into essential and inessential steps.

1 Introduction

The computational λ-calculus, called λ_c, was introduced by Moggi [Mog89, Mog91] as a meta-language to describe non functional effects in programming languages via an incremental approach. Much as for ordinary λ-calculus, the equational theory of λ_c can be modelled by the convertibility relation induced by a reduction relation. Building the reduction theory of λ_c is however quite challenging. A first attempt is in §6 of [Mog89], where the defined notion of reduction consists of six rules plus η. Proving confluence of this reduction relation revealed to be quite hard; it was studied in the context of call-by-need calculi, e.g. in [MOTW99, AFM+95] obtaining partial results, but a full proof has been achieved only recently in [Ham18].

Aiming at a logical analysis of the semantics of the untyped λ_c in terms of an intersection type assignment system, we proposed in [dT19] a simplified syntax, which is derived from Wadler's λ-calculus with monads, and defined reduction just by orientating the three monad laws in [Wad92, Wad95]. We dub λ_c^u our calculus, and $\longrightarrow_{\lambda\mathbf{C}}$ the reduction relation. This is the content of section 2 of the present note.

Although one can translate Moggi's syntax into ours, preserving and reflecting the respective reduction relations, the inverse translation just preserves conversion, so that confluence in our calculus cannot rest on the same property of the original λ_c, and the proof had to be reworked anew. We sketch the proof from [dT19] in section 3.

Confluence is not the only fundamental property of reduction in λ-calculi; further examples are standardization and the existence of normalizing strategies. Toward the study of these properties in the case of λ_c^u and $\longrightarrow_{\lambda\mathbf{C}}$, we explore here in section 4 factorization for our calculus by adapting results in [AFG19].

While the confluence proof is included in a revised version of [dT19] and has been submitted for publication, the factorization results are new.

2 Untyped λ_c-calculus

The syntax of the untyped computational λ-calculus, shortly λ_c^u, and its reduction relation as introduced in [dT19], are reported below:

Definition 2.1 (Terms of λ_c^u)**.** *The terms of the* untyped computational λ-calculus*, shortly λ_c^u, consist of two sorts of expressions:*

$$\begin{array}{llcll} Val: & V, W & ::= & x \mid \lambda x.M & \textit{(values)} \\ Com: & M, N & ::= & unit\, V \mid M \star V & \textit{(computations)} \end{array}$$

where x ranges over a denumerable set Var of variables. We set $Term = Val \cup Com$; $FV(V)$ and $FV(M)$ are the sets of free variables occurring in V and M respectively, and are defined in the obvious way. Terms are identified up to clash avoiding renaming of bound variables (α-congruence).

With respect to Moggi's λ_c-syntax, we do not have the *let* construct, which is considered as syntactical sugar for bind and abstraction:

$$let\ x = N\ in\ M \;\equiv\; N \star \lambda x.M$$

Notably we do not have application in the syntax, since it is definable (see below).

Definition 2.2 (Reduction)**.** *Define the following reduction relation $\mapsto_{\lambda \mathbf{C}} = \mapsto_{\beta_c} \cup \mapsto_{id} \cup \mapsto_{ass}$ over Com by:*

$$\begin{array}{lrcll} \beta_c) & unit\, V \star (\lambda x.M) & \mapsto & M[V/x] & \\ id) & M \star \lambda x.unit\, x & \mapsto & M & \\ ass) & (L \star \lambda x.M) \star \lambda y.N & \mapsto & L \star \lambda x.(M \star \lambda y.N) & \textit{for } x \notin FV(N) \end{array}$$

where $M[V/x]$ denotes the capture avoiding substitution of V for x in M. Finally define the relation $\longrightarrow_{\lambda \mathbf{C}}$ as the compatible closure of $\mapsto_{\lambda \mathbf{C}}$.

Rule β_c is reminiscent of the left unit law in [Wad95]; we call it β_c because it performs call-by-value β-contraction in λ_c^u. In fact, by reading $\star$ as postfix functional application and merging V into its trivial computation $unit\, V$, β_c is the same as β_v in [Plo75]. Now, let $V, W \in Val$ and $M, N \in Com$; then define:

$$\begin{array}{lcllcl} VW & \equiv & unit\, W \star V & MV & \equiv & M \star (\lambda z.unit\, V \star z) \\ VN & \equiv & N \star V & MN & \equiv & M \star (\lambda z.N \star z) \end{array}$$

where z is fresh. Then it is easy to see that, if $M \overset{*}{\longrightarrow}_{\lambda \mathbf{C}} unit\ (\lambda x.M')$ and $N \overset{*}{\longrightarrow}_{\lambda \mathbf{C}} unit\ V$ then $MN \overset{*}{\longrightarrow}_{\lambda \mathbf{C}} M'[V/x]$.

3 Confluence

Following a strategy used in case of call-by-need calculi with the *let* construct (see [AFM+95, MOTW99]), and more recently with the variant of call-by-value λ-calculus in [CG14], we split the proof in three steps, proving confluence of $\beta_c \cup id$ and *ass* separately, eventually combining these results by means of the commutativity of these relations.

In the first step we adapt the parallel reduction method, originally due to Tait and Martin Löf, and further developed by Takahashi [Tak95]. See e.g. the book [Ter03] ch. 10. Let's define the following relation $\Rrightarrow$:

Definition 3.1. *The relation $\Rrightarrow \subseteq Term \times Term$ is inductively defined by:*

i) $x \twoheadrightarrow x$
ii) $M \twoheadrightarrow N \Rightarrow \lambda x.M \twoheadrightarrow \lambda x.N$
iii) $V \twoheadrightarrow V' \Rightarrow \mathit{unit}\, V \twoheadrightarrow \mathit{unit}\, V'$
iv) $M \twoheadrightarrow M'$ *and* $V \twoheadrightarrow V' \Rightarrow M \star V \twoheadrightarrow M' \star V'$
v) $M \twoheadrightarrow M'$ *and* $V \twoheadrightarrow V' \Rightarrow \mathit{unit}\, V \star \lambda x.M \twoheadrightarrow M'[V'/x]$
vi) $M \twoheadrightarrow M' \Rightarrow M \star \lambda x.\mathit{unit}\, x \twoheadrightarrow M'$

By i) - iv) above, relation $\twoheadrightarrow$ is reflexive and coincides with its compatible closure. Also $\longrightarrow_{\beta_c, id} \subseteq \twoheadrightarrow$; intentionally, this is not the case w.r.t. the whole $\longrightarrow_{\lambda\mathbf{C}}$. Now, by means of Lemma 3.2 one easily proves that $\twoheadrightarrow \subseteq \xrightarrow{*}_{\beta_c, id}$.

Lemma 3.2. *For $M, M' \in \mathit{Com}$ and $V, V' \in \mathit{Val}$ and every variable x, if $M \twoheadrightarrow M'$ and $V \twoheadrightarrow V'$, then $M[V/x] \twoheadrightarrow M'[V'/x]$.*

The next step in the proof is to show that the relation $\twoheadrightarrow$ satisfies the *triangle property*:

$$TP: \qquad \forall P\, \exists P^*\, \forall Q.\; P \twoheadrightarrow Q \;\Rightarrow\; Q \twoheadrightarrow P^*$$

where $P, P^*, Q \in \mathit{Term}$. TP implies the *diamond property*, which for $\twoheadrightarrow$ is:

$$DP: \qquad \forall P, Q, R.\; P \twoheadrightarrow Q \;\&\; P \twoheadrightarrow R \;\Rightarrow\; \exists P'.\, Q \twoheadrightarrow P' \;\&\; R \twoheadrightarrow P'$$

In fact, if TP holds then we can take $P' \equiv P^*$ in DP, since the latter only depends on P. We then define P^* in terms of P as follows:

i) $x^* \equiv x$
ii) $(\lambda x.M)^* \equiv \lambda x.M^*$
iii) $(\mathit{unit}\, V)^* \equiv \mathit{unit}\, V^*$
iv) $(\mathit{unit}\, V \star \lambda x.M)^* \equiv M^*[V^*/x]$
v) $(M \star \lambda x.\mathit{unit}\, x)^* \equiv M^*$, if $M \not\equiv \mathit{unit}\, V$ for $V \in \mathit{Val}$
vi) $(M \star V)^* \equiv M^* \star V^*$, $M \not\equiv \mathit{unit}\, W$ for $W \in \mathit{Val}$ and $V \not\equiv \lambda x.\mathit{unit}\, x$

Lemma 3.3. *For all $P, Q \in \mathit{Term}$, if $P \twoheadrightarrow Q$ then $Q \twoheadrightarrow P^*$, namely $\twoheadrightarrow$ satisfies TP.*

According to [Bar84], Def. 3.1.11, a notion of reduction R is said to be *confluent* or *Church-Rosser*, shortly CR, if $\xrightarrow{*}_R$ satisfies DP; more explicitly for all $M, N, L \in \mathit{Com}$:

$$M \xrightarrow{*}_R N \;\&\; M \xrightarrow{*}_R L \Rightarrow \exists M' \in \mathit{Com}.\, N \xrightarrow{*}_R M' \;\&\; L \xrightarrow{*}_R M'$$

Corollary 3.4. *The notion of reduction $\beta_c \cup id$ is CR.*

To prove confluence of *ass* we use Newman Lemma (see [Bar84], Prop. 3.1.24). A notion of reduction R is *weakly Church-Rosser*, shortly WCR, if for all $M, N, L \in \mathit{Com}$:

$$M \longrightarrow_R N \;\&\; M \longrightarrow_R L \Rightarrow \exists M' \in \mathit{Com}.\, N \xrightarrow{*}_R M' \;\&\; L \xrightarrow{*}_R M'$$

Lemma 3.5. *The notion of reduction ass is WCR.*

Recall that a notion of reduction R is *strongly normalizing*, shortly SN, if there exists no infinite reduction $M \longrightarrow_R M_1 \longrightarrow_R M_2 \longrightarrow_R \cdots$ out of any $M \in \mathit{Com}$.

Lemma 3.6. *The notion of reduction ass is SN.*

Corollary 3.7. *The notion of reduction ass is CR.*

Proof. By Lem. 3.5, 3.6 and by Newman Lemma, stating that a notion of reduction which is *WCR* and *SN* is *CR*. □

Finally we show that $\longrightarrow_{\beta_c,id}$ and $\longrightarrow_{ass}$ commute. The following definitions are from [BN98], Def. 2.7.9. Relations $\longrightarrow_1$ and $\longrightarrow_2$ *strongly commute* if, for all M, N, L: $N\ {}_1\!\longleftarrow M \longrightarrow_2 L \Rightarrow \exists P.\ N \overset{=}{\longrightarrow}_2 P\ {}_1\!\overset{*}{\longleftarrow} L$ where $\overset{=}{\longrightarrow}_2$ is $\longrightarrow_2 \cup =$, namely at most one reduction step.

Lemma 3.8. *Reductions $\longrightarrow_{\beta_c,id}$ and $\longrightarrow_{ass}$ strongly commute, then commute.*

Proof. By Lemma 2.7.11 in [BN98], two strongly commuting relations commute, and commutativity is clearly symmetric; hence it suffices to show that

$$N\ {}_{\beta_c,id}\!\longleftarrow M \longrightarrow_{ass} L \Rightarrow \exists P \in \mathit{Com}.\ N \overset{=}{\longrightarrow}_{ass} P\ {}_{\beta_c,id}\!\overset{*}{\longleftarrow} L.$$

We can limit the cases to the critical pairs. For a full development see [dT19]. □

By the commutative union lemma (see [BN98], Lem. 2.7.10 and [Bar84], Prop. 3.3.5), if $\longrightarrow_{\beta_c,id}$ and $\longrightarrow_{ass}$ and are both *CR* (Cor. 3.4 and 3.7), and commute (Lem. 3.8) follows:

Theorem 3.9 (Confluence)**.** *The notion of reduction $\lambda\boldsymbol{C}$ is CR.*

4 Factorization

Specializing the definition of factorization in [AFG19], we say that an abstract reduction system $(\mathit{Term}, \longrightarrow)$ *factorizes* via $\longrightarrow_e, \longrightarrow_{\neg e}$ if $\longrightarrow\ =\ \longrightarrow_e \cup \longrightarrow_{\neg e}$ and for all $M, N \in \mathit{Term}$, $M \overset{*}{\longrightarrow} N$ implies that there exists $L \in \mathit{Term}$ such that $M \overset{*}{\longrightarrow}_e L \overset{*}{\longrightarrow}_{\neg e} N$. We abbreviate the last condition by $M \overset{*}{\longrightarrow}_e \cdot \overset{*}{\longrightarrow}_{\neg e} N$.

Now, we take $\longrightarrow\ =\ \longrightarrow_{\lambda\mathbf{C}}$ and construct the relations $\longrightarrow_e, \longrightarrow_{\neg e}$, called the *essential* and *inessential* in [AFG19], by closing $\mapsto_{\lambda\mathbf{C}}$ under two sorts of contexts:

$$\begin{array}{rrcl} \textit{Inessential contexts:} & \neg\mathcal{E} & ::= & \langle\cdot_{\mathsf{C}}\rangle \mid \mathit{unit}\ \lambda x.\neg\mathcal{E} \mid M \star \lambda x.\neg\mathcal{E} \mid \neg\mathcal{E} \star V \\ \textit{Essential contexts:} & \mathcal{E} & ::= & \langle\cdot_{\mathsf{C}}\rangle \mid \mathcal{E} \star V \end{array}$$

where the hole $\langle\cdot_{\mathsf{C}}\rangle$ can be filled by terms in *Com* only. Then $\longrightarrow_e$ and $\longrightarrow_{\neg e}$ are the least relations including $\mapsto_{\lambda\mathbf{C}}$ such that for all $M, N \in \mathit{Com}$, essential context $\mathcal{E}$ and inessential context $\neg\mathcal{E}$ it holds:

$$M \mapsto_{\lambda\mathbf{C}} N \Longrightarrow \mathcal{E}\langle M\rangle \longrightarrow_e \mathcal{E}\langle N\rangle \qquad \text{and} \qquad M \mapsto_{\lambda\mathbf{C}} N \Longrightarrow \neg\mathcal{E}\langle M\rangle \longrightarrow_{\neg e} \neg\mathcal{E}\langle N\rangle$$

We highlight that relations $\longrightarrow_e$ and $\longrightarrow_{\neg e}$ are actually not disjoint, as essential steps are also inessential.

The factorization property ensures that any finite reduction can be re-arranged into an essential reduction followed by some inessential steps. In our case, this corresponds to a weak head reduction, with the twist that in a bind expression the argument appears to the left of the function.

The key of the proof of the Factorization Theorem 4.4 is the construction of two further auxiliary relations $\Rightarrow_{\neg e}$ and $\Rightarrow_{\lambda\mathbf{C}}$, such that the conditions in Proposition 4.3 hold.

Definition 4.1 (Inessential parallel reduction)**.** *The relation $\Rightarrow_{\neg e} \subseteq \mathit{Term} \times \mathit{Term}$ is inductively defined by:*

i) $x \Rrightarrow_{\neg e} x$
ii) $M \Rrightarrow_{\lambda\mathbf{C}} N \Rightarrow \lambda x.M \Rrightarrow_{\neg e} \lambda x.N$
iii) $V \Rrightarrow_{\lambda\mathbf{C}} V' \Rightarrow \mathit{unit}\, V \Rrightarrow_{\neg e} \mathit{unit}\, V'$
iv) $M \Rrightarrow_{\neg e} M'$ *and* $V \Rrightarrow_{\neg e} V' \Rightarrow M \star V \Rrightarrow_{\neg e} M' \star V'$
v) $L \Rrightarrow_{\neg e} L'$ *and* $M \Rrightarrow_{\lambda\mathbf{C}} M'$ *and* $N \Rrightarrow_{\lambda\mathbf{C}} N' \Rightarrow (L \star \lambda x.M) \star \lambda y.N \Rrightarrow_{\neg e} L' \star \lambda x.(M' \star \lambda y.N')$

Definition 4.2 (Indexed parallel reduction)**.** *The relation* $\overset{n}{\Rrightarrow} \subseteq \mathit{Term} \times \mathit{Term}$ *is inductively defined by:*

i) $x \overset{0}{\Rrightarrow} x$

ii) $M \overset{n}{\Rrightarrow} N \Rightarrow \lambda x.M \overset{n}{\Rrightarrow} \lambda x.N$

iii) $V \overset{n}{\Rrightarrow} V' \Rightarrow \mathit{unit}\, V \overset{n}{\Rrightarrow} \mathit{unit}\, V'$

iv) $M \overset{n}{\Rrightarrow} M'$ *and* $V \overset{m}{\Rrightarrow} V' \Rightarrow M \star V \overset{n+m}{\Rrightarrow} M' \star V'$

v) $M \overset{n}{\Rrightarrow} M'$ *and* $V \overset{m}{\Rrightarrow} V' \Rightarrow \mathit{unit}\, V \star \lambda x.M \overset{n+|M'|_x \cdot m+1}{\Rrightarrow} M'[V'/x]$

vi) $M \overset{n}{\Rrightarrow} M' \Rightarrow M \star \lambda x.\mathit{unit}\, x \overset{n}{\Rrightarrow} M'$

vii) $L \overset{n}{\Rrightarrow} L'$ *and* $M \overset{m}{\Rrightarrow} M'$ *and* $N \overset{p}{\Rrightarrow} N' \Rightarrow (L \star \lambda x.M) \star \lambda y.N \overset{n+m+p}{\Rrightarrow} L' \star \lambda x.(M' \star \lambda y.N')$

where $|M|_x$ *is the number of free occurrences of* x *in* M*.*

Note that $\overset{0}{\Rrightarrow}$ is the identity relation on *Term*, $\overset{1}{\Rrightarrow}$ is $\longrightarrow_{\lambda\mathbf{C}}$ defined in 2.2, and $\overset{n}{\Rrightarrow} \subseteq \longrightarrow^n$. Define $\Rrightarrow_{\lambda\mathbf{C}} := \cup_{n\in\mathbb{N}} \overset{n}{\Rrightarrow}$. Observe that the above definition is essentially the same as that one of $\multimap\!\!\rightarrow$ in Def. 3.1, but for clause vii): adding the latter to $\multimap\!\!\rightarrow$ would break property *DP*, that indeed is not satisfied by $\Rrightarrow_{\lambda\mathbf{C}}$.

An abstract reduction system that satisfies the following conditions is called a *macro-step system* in [AFG19].

Proposition 4.3 ($\lambda\mathbf{C}$ Macro-step system)**.**
i) Merge*: if* $M \Rrightarrow_{\neg e} \cdot \longrightarrow_e M'$ *then* $M \Rrightarrow_{\lambda\mathbf{C}} M'$
ii) Indexed split*: if* $M \overset{n}{\Rrightarrow} M'$*, then* $M \Rrightarrow_{\neg e} M'$*, or* $n > 0$ *and* $M \longrightarrow_e \cdot \overset{n-1}{\Rrightarrow} M'$
iii) Split*: If* $M \Rrightarrow_{\lambda\mathbf{C}} M'$*, then* $M \overset{*}{\longrightarrow}_e \cdot \Rrightarrow_{\neg e} M'$*.*

Once we have established that $(\mathit{Term}, \longrightarrow_e \cup \longrightarrow_{\neg e})$ is a macro-step system with respect to $\Rrightarrow_{\lambda\mathbf{C}}$ and $\Rrightarrow_{\neg e}$. Since in [AFG19] is proved that every Macro-step system satisfies factorization, we have the following theorem.

Theorem 4.4 (Factorization)**.** *The reduction system* $(\mathit{Term}, \longrightarrow_{\lambda\mathbf{C}})$ *factorizes via* $\longrightarrow_e, \longrightarrow_{\neg e}$ *namely*

$$M \overset{*}{\longrightarrow}_{\lambda\mathbf{C}} M' \Rightarrow M \overset{*}{\longrightarrow}_e \cdot \overset{*}{\longrightarrow}_{\neg e} M'$$

References

[AFG19] Beniamino Accattoli, Claudia Faggian, and Giulio Guerrieri. Factorization and normalization, essentially. In *APLAS 2019: Programming Languages and Systems*, volume 11893 of *Lecture Notes in Computer Science*, page 159–180. Springer Verlag, 12 2019.

[AFM$^+$95] Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. In Ron K. Cytron and Peter Lee, editors, *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, pages 233–246. ACM Press, 1995.

[Bar84] H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, revised edition, 1984.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.

[CG14] Alberto Carraro and Giulio Guerrieri. A semantical and operational account of call-by-value solvability. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014*, volume 8412 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 2014.

[dT19] Ugo de'Liguoro and Riccardo Treglia. Intersection types for the computational lambda-calculus. *CoRR*, abs/1907.05706, 2019.

[Ham18] M. Hamana. Polymorphic rewrite rules: Confluence, type inference, and instance validation. In *Functional and Logic Programming - 14th International Symposium, FLOPS 2018, Nagoya, Japan, May 9-11, 2018, Proceedings*, volume 10818 of *Lecture Notes in Computer Science*, pages 99–115, 2018.

[Mog89] E. Moggi. Computational Lambda-calculus and Monads. In *Proceedings of Logic in Computer Science (LICS)*, pages 14–23, 1989.

[Mog91] E. Moggi. Notions of Computation and Monads. *Information and Computation*, 93:55–92, 1991.

[MOTW99] John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theor. Comput. Sci.*, 228(1-2):175–210, 1999.

[Plo75] G. D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[Tak95] M. Takahashi. Parallel reduction in lambda-calculus. *Information and Computation*, 118:120–127, 1995.

[Ter03] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[Wad92] P. Wadler. The essence of functional programming. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Albuquerque, New Mexico, USA, January 19-22, 1992*, pages 1–14, 1992.

[Wad95] P. Wadler. Monads for Functional Programming. In *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques-Tutorial Text*, volume 925 of *Lecture Notes in Computer Science*, pages 24–52. Springer-Verlag, 1995.

A Proof of Factorization

Lemma A.1. *(Substitutivity of $\overset{n}{\Longrightarrow}$) If $M \overset{n}{\Longrightarrow} M'$ and $V \overset{m}{\Longrightarrow} V'$ then $M[V/x] \overset{k}{\Longrightarrow} M'[V'/x]$ where $k = n + |M'|_x \cdot m$.*

Proof. Proof by induction on the structure of M. We will show just notable cases:
Application: This case occurs when the last rule applied is

$$\frac{N \overset{n_N}{\Longrightarrow} N' \qquad W \overset{n_W}{\Longrightarrow} W'}{M = N \star W \overset{n_N+n_W}{\Longrightarrow} N' \star W' = M'}$$

by i.h. $N[V/x] \overset{k_1}{\Longrightarrow} N'[V'/x]$ where $k_1 = n_N + |N'|_x \cdot m$

and $W[V/x] \overset{k_2}{\Longrightarrow} W'[V'/x]$ where $k_2 = n_W + |W'|_x \cdot m$
then

$$\frac{N[V/x] \overset{k_1}{\Longrightarrow} N'[V'/x] \qquad W[V/x] \overset{k_2}{\Longrightarrow} W'[V'/x]}{M[V/x] = N[V/x] \star W[V/x] \overset{k}{\Longrightarrow} N'[V'/x] \star W'[V'/x] = M'[V'/x]}$$

where $k = k_1+k_2 = n_N+|N'|_x\cdot m+n_W+|W'|_x\cdot m = n+|M'|_x\cdot m$, in fact $|M'|_x = |N'|_x+|W'|_x$.

β_c*−step*: This case occurs when the last step has the following shape:

$$\frac{W \overset{n_W}{\Longrightarrow} W' \qquad N \overset{n_N}{\Longrightarrow} N'}{M = \mathit{unit}\, W \star \lambda y.N \overset{n}{\Longrightarrow} N'[W'/y] = M'}$$

where $n = n_N + |N'|_y \cdot n_W + 1$.
Assuming wlog $x \neq y$, $|M'|_x = |N'[W'/y]|_x = |N'|_x + |N'|_y \cdot |W'|_x$

$$\begin{array}{c} M[V/x] = \mathit{unit}\, W[V/x] \star \lambda y.N[V/x] \\ M'[V'/x] = N'[V'/x][W'[V'/x]/y] \end{array}$$

By i.h. $N[V/x] \overset{k_1}{\Longrightarrow} N'[V'/x]$ where $k_1 = n_N + |N'|_x \cdot m$

$W[V/x] \overset{k_2}{\Longrightarrow} W'[V'/x]$ where $K_2 = n_W + |W'|_x \cdot m$, then $M[V/x] \overset{k}{\Longrightarrow} M'[V'/x]$ where

$$\begin{aligned} k &= k_1 + |N'|_y \cdot k_2 + 1 = \\ &= n_N + |N'|_x \cdot m + |N'|_y \cdot (n_W + |N'|_x \cdot m) + 1 = \\ &= n_N + |N'|_y \cdot n_W + 1 + |N'| \cdot m + |N'|_y \cdot |W'|_x \cdot m = \\ &= n + |M'|_x \cdot m \end{aligned}$$

id−step:

$$\frac{N \overset{n}{\Longrightarrow} N'}{M = N \star \lambda y.\mathit{unit}\, y \overset{n}{\Longrightarrow} N' = M'}$$

And $M[V/x] = N[V/x] \star \lambda y.\mathit{unit}\, y[V/x]$
By i. h. $N[V/x] \overset{k_1}{\Longrightarrow} N'[V'/x]$ where $k_1 = n + |N'|_x \cdot m$

$\lambda y.unit\, y[V/x] \overset{0}{\Rightarrow} \lambda y.unit\, y.$

$$\frac{N[V/x] \overset{k_1}{\Longrightarrow} N'[V'/x]}{M[V/x] = N[V/x] \star \lambda y.unit\, y[V/x] \overset{k}{\Rightarrow} N'[V'/x] = M'[V'/x]}$$

Then $k = k_1$.

ass−step: $M = (L \star \lambda y.N) \star \lambda z.P \overset{n}{\Rightarrow} L' \star \lambda y.(N' \star \lambda z.P)$ where $L \overset{n_L}{\Longrightarrow} L'$, $N \overset{n_N}{\Longrightarrow} N'$, $P \overset{n_P}{\Longrightarrow} P'$ and $n = N_l + n_N + n_P$.
by i.h.

$$L[V/x] \overset{k_1}{\Longrightarrow} L'[V'/x] \text{ where } k_1 = n_L + |L'|_x \cdot m$$
$$N[V/x] \overset{k_2}{\Longrightarrow} N'[V'/x] \text{ where } k_2 = n_N + |N'|_x \cdot m$$
$$P[V/x] \overset{k_3}{\Longrightarrow} P'[V'/x] \text{ where } k_3 = n_P + |P'|_x \cdot m$$

$$\frac{L[V/x] \overset{k_1}{\Longrightarrow} L'[V'/x] \qquad N[V/x] \overset{k_2}{\Longrightarrow} N'[V'/x] \qquad P[V/x] \overset{k_3}{\Longrightarrow} P'[V'/x]}{M[V/x] = (L[V/x] \star \lambda y.N[V/x]) \star \lambda z.P[V/x] \overset{k}{\Rightarrow} L'[V'/x] \star \lambda y.(N'[V'/x] \star \lambda z.P[V'/x])}$$

where $k = k_1 + k_2 + k_3 = n_L + |L'|_x \cdot m + n_N + |N'|_x \cdot m + n_P + |P'|_x \cdot m = n + |M'|_x \cdot m.$ □

Proposition A.2 ($\lambda\mathbf{C}$ Macro-step system)**.**
1. Merge*: if $M \Rightarrow_{\neg e} \cdot \longrightarrow_e M'$ then $M \Rightarrow_{\lambda\mathbf{C}} M'$*
2. Indexed split*: if $M \overset{n}{\Rightarrow} M'$, then $M \Rightarrow_{\neg e} M'$, or $n > 0$ and $M \longrightarrow_e \cdot \overset{n-1}{\Longrightarrow} M'$*
3. Split*: If $M \Rightarrow_{\lambda\mathbf{C}} M'$, then $M \longrightarrow_e^* \cdot \Rightarrow_{\neg e} M'$.*

Proof. 1.Merge: by structural induction on $M \Rightarrow_{\neg e} N$.
Following hypothesis, since $N \longrightarrow_e M'$, M' cannot be *unit* V for any $V \in Val$, then there exists an essential context $\mathcal{E}$, computations $\bar{N}$, $\bar{M}'$, such that $N = \mathcal{E}\langle\bar{N}\rangle \to_e \mathcal{E}\langle\bar{M}'\rangle = M'$.
Hence $N = \bar{N} \star \bar{V} = \to_e \bar{M}' \star \bar{V}' = M'$ and $M \Rightarrow_{\neg e} N$ is derived as follows

$$\frac{N_0 \Rightarrow_{\neg e} \bar{N} \qquad V_0 \Rightarrow_{\neg e} \bar{V}}{M = N_0 \star V_0 \Rightarrow_{\neg e} \bar{N} \star \bar{V} = N}$$

- if $\bar{N} \longrightarrow_e \bar{M}'$ then $M' = \bar{M}' \star \bar{V}$.
  The i.h. gives $N_0 \Rightarrow_{\lambda\mathbf{C}} \bar{M}'$, and $M \Rightarrow_{\lambda\mathbf{C}} M'$ is derived as follows

$$\frac{N_0 \Rightarrow_{\lambda\mathbf{C}} \bar{M}' \qquad V_0 \Rightarrow_{\lambda\mathbf{C}} \bar{V}}{M = N_0 \star V_0 \Rightarrow_{\lambda\mathbf{C}} \bar{M}' \star \bar{V} = M'}$$

- if $N \mapsto_{id} M'$ this means that $\bar{N} = M'$ and $\bar{V} = \lambda x.unit\, x$, and $M \Rightarrow_{\lambda\mathbf{C}} M'$ is derived as follows (since $\Rightarrow_{\neg e} \subseteq \Rightarrow_{\lambda\mathbf{C}}$)

$$\frac{N_0 \Rightarrow_{\lambda\mathbf{C}} \bar{N}}{M = N_0 \star \lambda x.unit\, x \Rightarrow_{\lambda\mathbf{C}} \bar{N} = M'}$$

- if $N \mapsto_{\beta_c} M'$ then $\bar{V} = \lambda x.L$ and $\bar{N} = \mathit{unit}\, W'$.
  By definition of $\Rightarrow_{\neg e}$ the step $V_0 \Rightarrow_{\neg e} \bar{V}$ ha the form
  $$\lambda x.L \Rightarrow_{\neg e} \lambda x.L' \text{ for some } L \text{ such that } L \Rightarrow_{\lambda\mathbf{C}} L'$$
  this means that $M \Rightarrow_{\lambda\mathbf{C}} M' = L[W'/x]$ following the next derivation
  $$\frac{L \Rightarrow_{\lambda\mathbf{C}} L' \qquad W \Rightarrow_{\lambda\mathbf{C}} W'}{M = \mathit{unit}\, W \star \lambda x.L \Rightarrow_{\lambda\mathbf{C}} L'[W'/x] = M'}$$
- if $N \mapsto_{ass} M'$ then $N = \bar{N} \star \bar{V} = (P \star \lambda x.Q) \star \bar{V}$ and $M' = \bar{M}' \star \bar{V}' = P \star \lambda x.(Q \star \bar{V})$.
  Since $N_0 \Rightarrow_{\neg e} \bar{N} = P \star \lambda x.Q$, it follows that N_0 has the shape $N_0 = P_0 \star \lambda x.Q_0$ where $P_0 \Rightarrow_{\neg e} P$ and $Q_0 \Rightarrow_{\lambda\mathbf{C}} Q$, then
  $$\frac{P_0 \Rightarrow_{\lambda\mathbf{C}} P \qquad Q_0 \Rightarrow_{\lambda\mathbf{C}} Q \qquad V_0 \Rightarrow_{\lambda\mathbf{C}} \bar{V}}{M = (P_0 \star \lambda x.Q_0) \star V_0 \Rightarrow_{\lambda\mathbf{C}} P \star \lambda x.(Q \star \bar{V}) = M'}$$

The associativity case follows similarly.

2. Indexed split: by induction on $M \overset{n}{\Longrightarrow} M'$. We will show just notable cases concerning to the reduction steps:
id−step: $M = N \star \lambda x.\mathit{unit}\, x \overset{n}{\Longrightarrow} N' = M'$. Then

$$\frac{N \overset{n}{\Longrightarrow} N'}{M = N \star \lambda x.\mathit{unit}\, x \overset{n}{\Longrightarrow} N' = M'}$$

by i.h. either $M \Rightarrow_{\neg e} M'$ (but there is no $\Rightarrow_{\neg e}$ rule that can occur) or $M \not\Rightarrow_{\neg e} M'$. This means that $M \not\Rightarrow_{\neg e} N'$ and by i.h. there exists N'' s.t. $M \to_e N'' \overset{n-1}{\Longrightarrow} N'$ so $M = N \star \lambda x.\mathit{unit}\, x \to_e N'' \star \lambda x.\mathit{unit}\, x \overset{n-1}{\Longrightarrow} N'$.

β_c−step: $M = \mathit{unit}\ V \star \lambda x.N \overset{k}{\Longrightarrow} N'[V'/x]$ where $k = n + |N'|_x \cdot m + 1$, where $N \overset{n}{\Longrightarrow} N'$ and $V \overset{m}{\Longrightarrow} V'$.
We have $M = \mathit{unit}\ V \star \lambda x.N \to_e N[V/x]$ and the substitutivity of $\overset{n}{\Longrightarrow}$ gives $M'' = N[V/x] \xRightarrow{n+|N'|_x \cdot m} N'[V'/x]$.

ass−step: If $M \overset{n}{\Longrightarrow} M'$ where $M = (L \star \lambda x.N) \star \lambda y.P$, $M' = L' \star \lambda x.(N' \star \lambda y.P')$ and $L \overset{n_L}{\Longrightarrow} L'$, $N \overset{n_N}{\Longrightarrow} N'$, $P \overset{n_P}{\Longrightarrow} P'$. There are two sub cases: either it is the case $M \Rightarrow_{\neg e} M'$, and $L \Rightarrow_{\neg e} L'$, $N \Rightarrow_{\neg e} N'$, $P \Rightarrow_{\neg e} P'$, then the claim holds.
Otherwise, if $M \not\Rightarrow_{\neg e} M'$, $L \not\Rightarrow_{\neg e} L'$ and $n_L > 0$ have to hold (otherwise $M \Rightarrow_{\neg e} M'$). By i.h. there exists $\bar{L}$ such that $L \longrightarrow_e \bar{L} \overset{n_L-1}{\Longrightarrow} L'$. So $M = (L \star \lambda x.N) \star \lambda y.P \longrightarrow_e (\bar{L} \star \lambda x.N) \star \lambda y.P \overset{n-1}{\Longrightarrow} M'$.

3. Split: if $M \Rightarrow_{\lambda\mathbf{C}} M'$ then there exists n such that $M \overset{n}{\Longrightarrow} M'$. By induction on n: by indexed split property just proved there are two cases:

1. $M \Rightarrow_{\neg e} M'$ and the statement is proved since $\xrightarrow{*}_e$ is reflexive.
2. $n > 0$ and there exists $\bar{M}$ such that $M \longrightarrow_e \bar{M} \xRightarrow{n-1} M'$. By i.h. applied to $\bar{M} \xRightarrow{n-1} M'$ there exists M'' such that $\bar{M} \longrightarrow_e M'' \Rightarrow_{\neg e} M'$ and so $M \xrightarrow{*}_e M'' \Rightarrow_{\neg e} M'$.

$\square$

Confluence in Lens Synthesis

Anders Miltner[1], Kathleen Fisher[2], Benjamin C. Pierce[3], David Walker[4], and Steve Zdancewic[5]

[1] Princeton University
amiltner@cs.princeton.edu
[2] Tufts University
kfisher@eecs.tufts.edu
[3] University of Pennsylvania
bcpierce@cis.upenn.edu
[4] Princeton University
dpw@cs.princeton.edu
[5] University of Pennsylvania
stevez@cis.upenn.edu

Abstract

A lens is a program that can be executed both forwards and backwards, from input to output and from output back to input again. Domain-specific languages for defining lenses have been developed to help users synchronize text files, and construct different "views" of databases, among other applications. Recent research has shown how string lenses can be synthesized from their types, which are pairs of regular expressions. However, guaranteeing that we can synthesize all possible lenses is quite tricky on these languages, due in large part to the many equivalences on regular expressions.

The proof that all string lenses are synthesizeable involves proving a confluence-like property, parameterized by a an additional binary relation R. We call this property R-confluence. In this model, standard confluence is the specific case where R is equality. In this paper, we show how existing techniques for demonstrating confluence do not work in the domain of R-confluence, and find that if the rewrite system is $=$-confluent and satisfies a commutativity property with R, then the system is R-confluent.

1 Introduction

Bidirectional transformations are pervasive in modern software systems, occuring as database views and view updaters, parsers and pretty-printers, data synchronization tools, and more. Instead of manually building the functions that comprise a bidirectional transformation, programmers can build them both "at once" using a bidirectional programming language. Bidirectional programming languages have been developed for creating view updaters [3], Linux configuration file editors [1], direct manipulation programming systems [11], and more [7, 5, 18]. Lenses are a particularly well-behaved class of bidirectional programs, where the underlying transformations are guaranteed to satisfy a number of "round-tripping" laws. Lens-based bidirectional programming languages often provide round-tripping guarantees through a set of typing rules; well-typed lens expressions are guaranteed to satisfy the round-tripping laws.

Optician [13], an extension of Boomerang, makes bidirectional programming easier by supporting synthesis of bidirectional string transformations. More specifically, it takes as input two regular expressions (R and S, which serve as the type of a Boomerang lens) and a set of examples specifying input-output behavior, and synthesizes a well-typed lens between the languages of those regular expressions. For brevity, we will not provide formal definitions for some aspects of Optician; the interested reader can find such definitions in the original Optician paper [13]. Furthermore, examples detailing the use of synthesized lenses in practice (which we also elide for space) can be found in that paper and follow-up work [8, 14].

To indicate a lens l is well typed, and converts between the languages of R and S, we write $l : R \Leftrightarrow S$. For synthesis, given R and S, we must find a lens l such that $l : R \Leftrightarrow S$. In the context of lens synthesis, there are three sorts of lens typing rules: syntax-directed rules, composition, and type equivalence.

For syntax-directed rules, the syntax for the types closely mirrors the syntax of the expressions. As an example, consider the following rule for disjunction:

$$\textsc{Or Lens}\quad \frac{\begin{array}{cc} l_1 : R_1 \Leftrightarrow S_1 & l_2 : R_2 \Leftrightarrow S_2 \\ \mathcal{L}(R_1) \cap \mathcal{L}(R_2) = \emptyset & \mathcal{L}(S_1) \cap \mathcal{L}(S_2) = \emptyset \end{array}}{or(l_1, l_2) : R_1 \mid R_2 \Leftrightarrow S_1 \mid S_2}$$

Consider finding a lens of type $R_1 \mid R_2 \Leftrightarrow S_1 \mid S_2$. With only syntax-directed rules, the only lens that can be well-typed would be an or lens.

Composition sequentially composes two lenses.

$$\textsc{Composition}\quad \frac{l_1 : R_1 \Leftrightarrow R_2 \qquad l_2 : R_2 \Leftrightarrow R_3}{l_1 \,;\, l_2 : R_1 \Leftrightarrow R_3}$$

Composition is difficult in the context of lens synthesis. If trying to synthesize a composition lens, one has to pull the central regular expression "out of thin air."

The last typing rule is type-equivalence. If two regular expressions are star-semiring equivalent to the type of a lens, those equivalent regular expressions also serve as the type of the lens.

$$\textsc{Type Equivalence}\quad \frac{l : R \Leftrightarrow S \qquad R \equiv^s R' \qquad S \equiv^s S'}{l : R' \Leftrightarrow S'}$$

This rule is difficult in the context of synthesis, as it forces a search through equivalent regular expressions. This rule can also be applied at any point in the derivation, which makes the search even harder.

To address the difficulties with composition and type-equivalence rules, we synthesize lenses in an alternative language of disjunctive normal form (DNF) lenses. DNF lenses are in pseudonormal form, containing no composition operator, and so their synthesis never needs to pull regular expressions "out of thin air." The types of DNF lenses are pairs of regular expressions in a pseudonormal form, DNF regular expressions. Because DNF regular expressions are in a pseudonormal form, fewer equivalent regular expressions need to be searched through.

In our search algorithm, we only search through equivalent regular expressions once, before processing any syntax directed-rules. We formalize this in the typing of DNF regular expressions by only permitting the application of type-equivalence once, after all syntactic rules have been applied. This is enforced by having two typing judgements: one for the "rewriteless" type of the lens (meaning no type-equivalence rules were applied) and one of the "full" type of the lens. If $dl \mathrel{\tilde{:}} DR \Leftrightarrow DS$, then dl is a DNF lens of rewriteless type $DR \Leftrightarrow DS$. If $dl : DR \Leftrightarrow DS$, then dl is a DNF lens of full type $DR \Leftrightarrow DS$. The following rule is used to get the full type of a DNF lens from the rewriteless type.

$$\textsc{Rewrite DNF Regex Lens}\quad \frac{DR' \rightarrow^* DR \qquad DS' \rightarrow^* DS \qquad dl \mathrel{\tilde{:}} DR \Leftrightarrow DS}{dl : DR' \Leftrightarrow DS'}$$

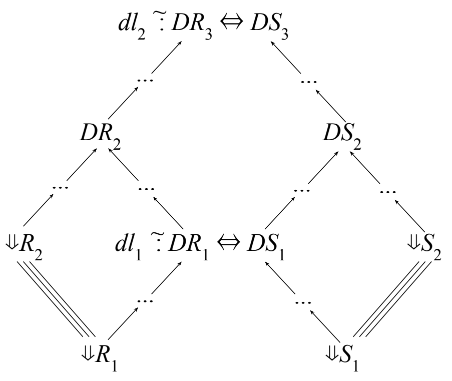

$dl_2 \,\tilde{:}\, DR_3 \Leftrightarrow DS_3$
DR_2
DS_2
$\Downarrow R_2$
$dl_1 \,\tilde{:}\, DR_1 \Leftrightarrow DS_1$
$\Downarrow S_2$
$\Downarrow R_1$
$\Downarrow S_1$

Figure 1: Diagram constructing a well-typed DNF lens between $\Downarrow R_2$ and $\Downarrow S_2$. Single lined arrows indicate rewrites ($\rightarrow$), and triple lines indicate equivalence ($\equiv_\rightarrow$).

Note that instead of using directionless equivalences, Rewrite DNF Regex Lens uses directioned rewrites. The relationship between the two is formalized by the following theorem:

Theorem 1. If $R \equiv^s S$, then $\Downarrow R \equiv_\rightarrow \Downarrow S$, where $\Downarrow R$ and $\Downarrow S$ are R and S in DNF form (respectively), and $\equiv_\rightarrow$ is the reflexive, transitive, and symmetric closure of $\rightarrow$.

Proving that our search procedure can generate any lens reduces to proving DNF lenses complete with respect to our standard lens language. In particular, we wish the prove:

Theorem 2. If $l : R \Leftrightarrow S$, then there exists a DNF lens, dl, such that $dl : \Downarrow R \Leftrightarrow \Downarrow S$ and the semantics of l and dl are equivalent.

We prove this propety by induction on the structure of the typing derivation. Particular difficulty lies in the lens equivalence rule. We begin this case below:

$$\frac{l : R_1 \Leftrightarrow S_1 \qquad R_1 \equiv^s R_2 \qquad S_1 \equiv^s S_2}{l : R_2 \Leftrightarrow S_2}$$

By induction assumption, there exists $dl : \Downarrow R_1 \Leftrightarrow \Downarrow S_1$, where the semantics of dl are equivalent to those of l. By inversion on the derivation of $dl : \Downarrow R_1 \Leftrightarrow \Downarrow S_1$, there exists DR_1 and DS_1 such that:

$$\frac{\Downarrow R_1 \rightarrow^* DR_1 \qquad \Downarrow S_1 \rightarrow^* DS_1 \qquad dl \,\tilde{:}\, DR_1 \Leftrightarrow DS_1}{dl : \Downarrow R_1 \Leftrightarrow \Downarrow S_1}$$

To complete this case, we need to find a DNF lens $dl' : \Downarrow R_2 \Leftrightarrow \Downarrow S_2$ with equivalent semantics to l.

We first show that there exist DR_2 and DS_2 such that $\Downarrow R_2 \rightarrow^* DR_2$ and $\Downarrow DR_1 \rightarrow^* DR_2$ and $\Downarrow S_2 \rightarrow^* DS_2$ and $\Downarrow DS_1 \rightarrow^* DS_2$. To do this, we first prove that $\rightarrow$ is confluent. By Theorem 1, we know that $\Downarrow R_1 \equiv_\rightarrow \Downarrow R_2$, so confluence implies the existance of such a DR_2 and DS_2.

After this, we have $dl \,\tilde{:}\, DR_1 \Leftrightarrow DS_1$ and $DR_1 \rightarrow^* DR_2$ and $DS_1 \rightarrow^* DS_2$. If we can prove a confluence-like property that would show the existance of some $dl' \,\tilde{:}\, DR_3 \Leftrightarrow DS_3$, where $DR_2 \rightarrow^* DR_3$ and $DS_2 \rightarrow^* DS_3$, we would be done. This property is R-confluence for a properly chosen R (which we describe in §2). This case is diagrammed in Figure 1.

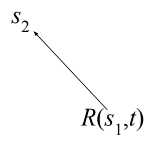


Figure 2: Rewrite system that satisfies the R-diamond property, but is not R-confluent.

2 R-Confluence Formulation

Let S be an underlying set, and $\rightarrow$ and R be binary relations. The rewrite system $(S, \rightarrow)$ is R-confluent, if for all $s_1, s_2 \in S$, if $R(s_1, s_2)$, $s_1 \rightarrow^* s_1'$, and $s_2 \rightarrow^* s_2'$, then there exist s_1'' and s_2'' such that $R(s_1'', s_2'')$, $s_1' \rightarrow^* s_1''$, and $s_2 \rightarrow^* s_2''$.

In the context of lens synthesis, S is the set of DNF REs, the rewrites are $\rightarrow^*$, and given a DNF lens dl, $R_{dl}(DR, DS)$ is true if there exists a DNF lens dl' such that $dl' \,\tilde{:}\, DR \Leftrightarrow DS$ and dl is equivalent to dl'.

Now that R-confluence has been formally defined, we can ask ourselves: "What is a good approach to proving R-confluence?" One approach is to prove that our rewrite system is locally confluent, which is equivalent to =-confluence in a terminating system [6]. Unfortunately, our rewrites are not terminating, so this approach does not work.

An approach pioneered by Tait and Martin-Löf [2] still works in non-terminating systems. This approach uses the *diamond property*: A rewrite system $(S, \rightarrow)$ satisfies the diamond property if $s_1 \rightarrow s_2$ and $s_1 \rightarrow s_3$ implies that there exists s_4 such that $s_2 \rightarrow s_4$, and $s_2 \rightarrow s_4$. If a rewrite system satisfies the diamond property, then it is also confluent. Unfortunately, this approach does not work, as the parameterized version of the diamond property does not imply R-confluence.

3 Proving (S,$\rightarrow^*$) R-Confluent

In the Tait and Martin-Löf approach to proving $(S, \rightarrow)$ confluent, one must first prove $(S, \rightarrow)$ satisfies the diamond property. Consider a parameterized property analogous to the diamond property, the *R-diamond property*: A rewrite system $(S, \rightarrow)$ satisfies the R-diamond property if $s_1 \rightarrow s_2$ and $t_1 \rightarrow t_2$ and $R(s_1, t_1)$ implies that there exists s_3, t_3 such that $s_2 \rightarrow s_3$ and $t_2 \rightarrow t_3$ and $R(s_3, t_3)$.

However, satisfying the R-diamond property is not sufficient for R-confluence. Consider the simple rewrite system shown in Figure 2. In this rewrite system, there are 3 elements, s_1, s_2, and t. In this setup, $R(s_1, t)$ and $s_1 \rightarrow s_2$. R-confluence requires some s_3, t' such that $s_2 \rightarrow^* s_3$ and $t \rightarrow^* t'$ and $R(s_3, t')$, but no such values exist.

To get around this issue, we require a different set of properties.

1. $(S, \rightarrow)$ must be =-confluent.

2. R must be a bisimilulation relation for $(S, \rightarrow^*)$. In other words if $R(s_1, t_1)$, and $s_1 \rightarrow^* s_2$, then there exists t_2 such that $t_1 \rightarrow^* t_2$ and $R(s_2, t_2)$; and if $R(s_1, t_1)$, and $t_1 \rightarrow^* t_2$, then there exists s_2 such that $s_1 \rightarrow^* s_2$ and $R(s_2, t_2)$.

Theorem 3. Let $(S, \rightarrow)$ be =-confluent, and R be a bisimilulation relation for $(S, \rightarrow^*)$. If $R(s_1, t_1)$ and $s_1 \rightarrow^* s_2$ and $t_1 \rightarrow^* t_2$, then there exists s_3, t_3 such that $s_2 \rightarrow^* s_3$ and $t_2 \rightarrow t_3$ and $R(s_3, t_3)$.

$$\begin{tikzcd}
 & & R(s_5,t_5) & & & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & & & \\
s_3 \arrow[ur, dash] & & & & R(s_4,t_4) \arrow[ul, dash] & & \\
 & & & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
 & & s_2 \arrow[uull] \arrow[ur, dash] & & & & t_2 \arrow[ul, dash] \\
 & & & \cdots \arrow[ul] & & \cdots \arrow[ur] & \\
 & & & & R(s_1,t_1) \arrow[ul, dash] \arrow[ur, dash] & &
\end{tikzcd}$$

Figure 3: Diagram showing how we prove the inductive case of R-confluence of the transitive closure of a rewrite system. $R(s_4,t_4)$ comes from the inductive hypothesis. The existance of s_5 is guaranteed through =-confluence. $R(s_5,t_5)$ is guaranteed because R is a bisimulation relation.

Proof. By induction length of the reduction of $s_1 \to^* s_2$

Case 1 (Base Case). Let $R(s_1,t_1)$ and $t_1 \to^* t_2$. As R is a bisimulation relation for $(S, \to^*)$, there exists s_2 such that $s_1 \to^* s_2$ and $R(s_2,t_2)$, as desired.

Case 2 (Inductive Case). Let $R(s_1,t_1)$ and $s_1 \to^* s_2$ and $s_2 \to s_3$ and $t_1 \to^* t_2$. By the induction hypothesis, there exists s_4, t_4 such that $R(s_4,t_4)$ and $s_2 \to^* s_4$ and $t_2 \to t_4$.

Because $(S, \to)$ is =-confluent, there exists s_5 such that $s_3 \to^* s_5$ and $s_4 \to^* s_5$. As $(S, \to^*)$ is a bisimilulation relation on $(S, \to^*)$, there exists t_5 such that $t_4 \to^* t_5$ and $R(s_5,t_5)$, as desired. This case is diagrammed in Figure 3.

□

4 Related Work

The concept of R-confluence is related to the notion of *confluence modulo* $\sim$ [6]. The definition of confluent modulo $\sim$ is almost the same as $\sim$-confluence, the only difference is that confluence modulo $\sim$ requires $\sim$ to an equivalence relation. Conditions that suffice to prove a rewrite system confluent modulo $\sim$ are not generally sufficient to prove R-confluence (and vice versa). Furthermore, our bisimulation relations are closely related to local coherence modulo $\sim$.

Bisimulation relations come from concurrency theory [15], but a related notion, commuting rewrites [16], appear in the confluence literature. We require a single rewrite of R to commute with an arbitrary number of rewrites of $\to$, which commuting rewrites do not express.

The full proof of completeness is contained in the appendix of the full version of the original optician paper [12]. The original proof of R-confluence for the transitive closure of $\to$ required additional assumptions. These unnecessary assumptions have been identified and removed in this paper. Future work used the proof of completeness over our lens language to show that quotient bijective lenses are also synthesizeable [8]. Lastly, while synthesizeability was not proven for symmetric lenses [14], such a proof would likely have proven R-confluence in a similar manner.

This work continues a trend in making programming easier through synthesis [4]. While synthesis is one approach to make bidirectional programming easier, it is not the only approach. Work has gone into building lenses without requiring a point-free combinator style [10]. Other work has found applicative [9] and monadic [17] approaches to compositionally building lenses.

References

[1] Augeas - A configuration API. http://augeas.net/index.html.

[2] H. P. Barendregt. *The Lambda Calculus Its Syntax and Semantics*, volume 103. North Holland, revised edition, 1984. http://www.cs.ru.nl/ henk/Personal Webpage.

[3] Aaron Bohannon, Jeffrey A. Vaughan, and Benjamin C. Pierce. Relational lenses: A language for updateable views. In *Principles of Database Systems (PODS)*, 2006. Extended version available as University of Pennsylvania technical report MS-CIS-05-27.

[4] Sumit Gulwani, Alex Polozov, and Rishabh Singh. *Program Synthesis*, volume 4. NOW, August 2017.

[5] Soichiro Hidaka, Zhenjiang Hu, Kazuhiro Inaba, Hiroyuki Kato, Kazutaka Matsuda, and Keisuke Nakano. Bidirectionalizing graph transformations. In *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010, Baltimore, Maryland, USA, September 27-29, 2010*, pages 205–216, 2010.

[6] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, October 1980.

[7] Hsiang-Shang Ko, Tao Zan, and Zhenjiang Hu. BiGUL: A formally verified core language for putback-based bidirectional programming. In *Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 61–72, 2016.

[8] Solomon Maina, Anders Miltner, Kathleen Fisher, Benjamin C. Pierce, David Walker, and Steve Zdancewic. Synthesizing quotient lenses. *Proc. ACM Program. Lang.*, 2(ICFP), July 2018.

[9] Kazutaka Matsuda and Meng Wang. Applicative bidirectional programming with lenses. *SIGPLAN Not.*, 50(9):62–74, August 2015.

[10] Kazutaka Matsuda and Meng Wang. Hobit: Programming lenses without using lens combinators. In Amal Ahmed, editor, *Programming Languages and Systems*, pages 31–59, Cham, 2018. Springer International Publishing.

[11] Mikaël Mayer, Viktor Kuncak, and Ravi Chugh. Bidirectional evaluation with direct manipulation. *Proc. ACM Program. Lang.*, 2(OOPSLA), October 2018.

[12] Anders Miltner, Kathleen Fisher, Benjamin C. Pierce, David Walker, and Steve Zdancewic. Synthesizing bijective lenses, 2017. https://arxiv.org/abs/1710.03248.

[13] Anders Miltner, Kathleen Fisher, Benjamin C. Pierce, David Walker, and Steve Zdancewic. Synthesizing bijective lenses. In *Proceedings of the 45th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL 2018, 2018.

[14] Anders Miltner, Solomon Maina, Kathleen Fisher, Benjamin C. Pierce, David Walker, and Steve Zdancewic. Synthesizing symmetric lenses. *Proc. ACM Program. Lang.*, 3(ICFP), July 2019.

[15] Davide Sangiorgi. *Introduction to Bisimulation and Coinduction.* Cambridge University Press, USA, 2011.

[16] Yoshihito Toyama. Commutativity of term rewriting systems. *Programming of future generation computers II*, pages 393–407, 1988.

[17] Li-yao Xia, Dominic Orchard, and Meng Wang. Composing bidirectional programs monadically. In Luís Caires, editor, *Programming Languages and Systems*, pages 147–175, Cham, 2019. Springer International Publishing.

[18] Zirun Zhu, Hsiang-Shang Ko, Pedro Martins, João Saraiva, and Zhenjiang Hu. Biyacc: Roll your parser and reflective printer into one. In *Proceedings of the 4th International Workshop on Bidirectional Transformations co-located with Software Technologies: Applications and Foundations, STAF 2015, L'Aquila, Italy, July 24, 2015.*, pages 43–50, 2015.

Coherent Confluence in Modal $\mathfrak{n}$-Kleene Algebras

Cameron Calk

Laboratoire de l'École Polytechnique (LIX), Paris, France
cameron.calk@inria.fr

Abstract

This work concerns the algebraic formalisation of coherence and confluence. The Church-Rosser theorem states that the confluence of branchings and the confluence of zig-zags are equivalent properties. On the one hand, the coherent version of this theorem has been formulated in the language of higher dimensional globular categories. On the other hand, the Church-Rosser theorem has also been formulated using the structure of modal Kleene algebra. This work provides an algebraic formulation of the coherent Church-Rosser theorem. To this end, we introduce the structure of $\mathfrak{n}$-dimensional modal Kleene algebra satisfying globularity properties, providing a higher dimensional generalisation of modal Kleene algebra. In this setting, we give abstract definitions of coherent rewriting properties and, via modal operators, present novel proofs of the Church-Rosser theorem with higher-dimensional witnesses.

1 Introduction

In rewriting theory, a central theme is that of completing certain *branching shapes*, *i.e.* (local) branchings or zig-zags, with confluences, thus obtaining *confluence diagrams*. This is classically described with relations, but has more recently been formulated in the higher dimensional setting of polygraphs [2, 5]. While this approach is adapted to the study of string rewriting systems, it also provides a natural setting for the study of *coherence* [5] in abstract rewriting systems. Furthermore, a point-free algebraic formalisation of confluence is given in [7]. This abstracts the relational approach to the setting of Kleene algebras, providing a more general framework for the study of confluence. In this work, we combine these two approaches by introducing the structure of $\mathfrak{n}$-Kleene algebra, a natural setting for abstract, higher dimensional confluence proofs, and illustrate its use with the example of the Church-Rosser theorem.

In the relational setting, confluence is characterised, for an abstract rewriting system $\rightarrow$, *i.e.* a binary relation on some set, by the inclusion

$$\overset{*}{\leftarrow} \cdot \overset{*}{\rightarrow} \quad \subseteq \quad \overset{*}{\rightarrow} \cdot \overset{*}{\leftarrow},$$

where $\overset{*}{\rightarrow}$ denotes the reflexive, transitive closure of $\rightarrow$, $\leftarrow$ denotes its converse and $\cdot$ denotes relational composition. The Church-Rosser property is characterised by the inclusion

$$\overset{*}{\leftrightarrow} \quad \subseteq \quad \overset{*}{\rightarrow} \cdot \overset{*}{\leftarrow},$$

where $\overset{*}{\leftrightarrow} = (\leftarrow \cup \rightarrow)^*$ is the symmetric, reflexive, transitive closure of $\rightarrow$. The Church-Rosser theorem, which states that these two properties are equivalent, can be formulated with similar expressions in Kleene algebra using the *star* operation, an abstraction of the notion of reflexive, transitive closure, see Section 2.

The coherent formulation of this theorem takes place in the context of polygraphs. Roughly, it states that if there exists a set Γ of higher dimensional cells (with a globular shape) such that every branching can be completed to a confluence diagram filled by some element of Γ, then every zig-zag may be completed with a Church-Rosser diagram filled by a composition of elements in Γ, see [2]. In diagrams, these statements are respectively represented by:

$$\begin{tikzcd}
 & u \arrow[dl, "f"'] \arrow[dr, "g"] \arrow[d, phantom, "\Downarrow\alpha"] & \\
u_1 \arrow[dr, dashed, "f'"'] & {} & u_2 \arrow[dl, dashed, "g'"] \\
 & u' &
\end{tikzcd}
\qquad
\begin{tikzcd}
u_1 \arrow[rr, leftrightarrow, "h"] \arrow[dr, dashed, "f'"'] & \Downarrow\beta & u_2 \arrow[dl, dashed, "g'"] \\
 & u' &
\end{tikzcd}$$

where $\alpha \in \Gamma$ and β is a composition of elements of Γ. This is a coherence result akin to *e.g.* Mac Lane's theorem for monoidal categories, since local data, *i.e.* existence of confluence diagrams, is constructively used to prove a global result, *i.e.* existence of Church-Rosser diagrams.

In this abstract, we recall the structure of modal n-Kleene algebra from [2] and illustrate its use in the algebraic formalisation of coherence results by formulating and giving a proof sketch of the coherent Church-Rosser theorem. In Section 2 we provide background material, defining modal Kleene algebras as in [3] and Γ-coherence properties, an extension of constructions from [4]. Next, we introduce the novel structure of modal n-Kleene algebra and provide axioms for globularity in Section 3. Finally, in Section 4, we formalise the notion of coherence, state the Church-Rosser theorem in the setting of n-Kleene algebras and give a short proof sketch.

2 Modal Kleene Algebras and Coherence

Modal Kleene Algebras, [3]. Recall that a *dioid* is a semiring $(S, +, \cdot, 0, 1)$ in which addition is idempotent, *i.e.* for all $x \in S$, we have $x + x = x$. In this case, the natural partial order on S is defined by $x \leq y \;\Leftrightarrow\; x + y = y$. A *domain semiring* is a dioid $(S, +, \cdot, 0, 1)$ equipped with a *domain operation* $d : S \longrightarrow S$ that satisfies, for all $x, y \in S$, the following axioms :

$$x \leq d(x)x \quad (1) \qquad d(xy) = d(xd(y)) \quad (2) \qquad d(x) \leq 1 \quad (3)$$

$$d(0) = 0 \quad (4) \qquad d(x + y) = d(x) + d(y) \quad (5)$$

where juxtaposition indicates multiplication. These axioms characterise domain in the case of relational algebras. For a more detailed account, see [3]. In particular, they imply that $S_d := d(S)$ forms a distributive lattice with the induced operations $+$ as join and $\cdot$ as meet, bounded by 0 and 1.

The *opposite* of a semiring S, in which the order of multiplication has been reversed, is denoted by S^{op}. A *codomain semiring* is a semiring equipped with a map $r : S \longrightarrow S$ such that (S^{op}, r) is a domain semiring. A *modal semiring* is both a domain semiring and a codomain semiring, with additional axioms $d \circ r = r$ and $r \circ d = d$, which imply that $S_d = S_r$. We include these definitions for use in Section 3, in which the term *modal* will be justified.

A *(modal) Kleene algebra* is a (modal) semiring K equipped with an operator $(-)^* : K \longrightarrow K$ called *Kleene star*. It satisfies, for all $x, y, z \in K$, the *unfold* and *induction* axioms

$$1 + xx^* \leq x^* \qquad 1 + x^*x \leq x^* \tag{6}$$

$$z + xy \leq y \Rightarrow x^*z \leq y \qquad z + yx \leq y \Rightarrow zx^* \leq y \tag{7}$$

In a Kleene algebra K, the Church-Rosser theorem is expressed as follows: for elements $x, y \in K$, the following are equivalent:

$$x^*y^* \leq y^*x^* \qquad (x + y)^* \leq y^*x^* \tag{8}$$

(x and y semi-commute) (x and y are Church-Rosser.)

Γ-Coherence Properties. Recall that an n-polygraph P is a data consisting of sets $(P_k)_{0 \leq k \leq n}$ and *source* and *target* maps $(s_k, t_k : P_{k+1} \longrightarrow P_k^*)_{0 \leq k < n}$ satisfying globularity conditions, where P_k^* is the free k-category generated by the globular set $(P_0, P_1^*, \ldots, P_{k-1}^*, P_k)$. A *cellular extension* of P is a set Γ and (globular) attaching maps $s_n, t_n : \Gamma \to P_n^\top$, where $P_n^\top$ is the free $(n, n-1)$-category generated by P. See [5] for a more detailed account of polygraphs and their use in rewriting theory.

Given an n-polygraph P and a cellular extension Γ, we say that P is Γ-confluent (*resp.* Γ-Church-Rosser) if, for every branching $(f, g) \in (P_n^*)^2$ of n-cells (*resp.* every zig-zag $h \in P_n^\top$), there exist a confluence (f', g') and a $(n+1)$-cell $\alpha \in \Gamma$ (*resp.* $\alpha \in P_n^\top(\Gamma)$, the free $(n+1, n-1)$-category generated by (P, Γ)) such that

$$\begin{tikzcd} & u \arrow[dr, "g"] \arrow[dd, phantom, "\Downarrow \alpha"] & \\ u_1 \arrow[ur, "f^-"] \arrow[dr, "f'"'] & & v_1 \\ & u' \arrow[ur, "(g')^-"'] & \end{tikzcd} \qquad \begin{tikzcd} u \arrow[rr, "h"] \arrow[dr, "f'"'] & \Downarrow \alpha & v \\ & u' \arrow[ur, "g'^-"'] & \end{tikzcd}$$

$$\alpha : f^- \star_{n-1} g \to f' \star_{n-1} (g')^- \qquad (\textit{resp.}\ \ \alpha : f \to f' \star_{n-1} g'^-),$$

where, for n-cells f_1, f_2 of P, $f_1 \star_{n-1} f_2$ denotes their composition with respect to the $(n-1)$-dimensional target (*resp.* source) of f_1 (*resp.* f_2). Readers familiar with polygraphic rewriting may notice that the filling cells are usually oriented from left to right, rather than from top to bottom. We choose this orientation to better reflect the approach used in the setting of Kleene algebras, in which branching shapes, such as x^*y^* and $(x+y)^*$, are related to confluence shapes y^*x^*, as recalled in (8). This will equally be the case in n-Kleene algebra, see Section 4.

The coherent Church-Rosser theorem [2] is formulated as follows: given P a n-polygraph, and Γ a cellular extension of P, if P is Γ-confluent, then P is Γ-Church-Rosser.

3 Higher Dimensional Modal Kleene Algebra

n-Dioids. For $n \geq 1$, an *n-dioid* is a structure $(S, +, 0, \odot_i, 1_i)_{0 \leq i < n}$ such that $(S, +, 0, \odot_i, 1_i)$ is a dioid for $0 \leq i < n$, the *lax interchange change laws* hold, and units are idempotent with respect to lower dimensional multiplications, *i.e.*

$$(A \odot_j A') \odot_i (B \odot_j B') \leq (A \odot_i B) \odot_j (A' \odot_i B') \quad \text{and} \quad 1_j \odot_i 1_j = 1_j, \tag{9}$$

for all $A, A', B, B' \in S$ and $0 \leq i < j < n$. The opposite n-semiring of S, denoted S^{op}, is that in which the order of every multiplication operation has been reversed.

Globular n-Semirings with Domains. A *domain n-semiring* is an n-dioid $(S, +, 0, \odot_i, 1_i)_{0 \leq i < n}$ equipped with n domain maps $(d_i : S \to S)_{0 \leq i < n}$, such that $(S, +, 0, \odot_i, 1_i, d_i)$ is a domain semiring and $d_{i+1} \circ d_i = d_i$ for all $0 \leq i < n-1$. A n-semiring with codomains is equipped with maps $(r_i : S \to S)_{0 \leq i < n}$ such that S^{op} is a domain n-semiring with respect to the $(r_i)_{0 \leq i < n}$.

A *modal n-semiring* is an n-semiring with domains and codomains, in which the coherence conditions $d_i \circ r_i = r_i$ and $r_i \circ d_i = d_i$ hold for all $0 \leq i < n$. Given S a modal n-semiring, we define *forward* and *backward i-diamond* operators defined via (co-)domain operators in each dimension, as in [3]. For any $0 \leq i < n$, $A \in S$ and $\phi \in S_i$, we define

$$|A\rangle_i(\phi) = d_i(A \odot_i \phi), \quad \text{and} \quad \langle A|_i(\phi) = r_i(\phi \odot_i A). \tag{10}$$

These are modal operators on the distributive lattice S_i in the sense of [6]. A modal semiring S is called *globular* if the following *globular relations* hold for $0 \leq i < j < n$ and $A, B \in K$:

$$d_i \circ d_j = d_i \text{ and } d_i \circ r_j = d_i, \quad (11) \qquad d_j(A \odot_i B) = d_j(A) \odot_i d_j(B), \quad (13)$$

$$r_i \circ d_j = r_i, \text{ and } r_i \circ r_j = r_i, \quad (12) \qquad r_j(A \odot_i B) = r_j(A) \odot_i r_j(B). \quad (14)$$

Modal n-Kleene Algebra. A *n-Kleene algebra* is a n-semiring K equipped with maps $(-)^{*_i} : K \to K$

such that $(K, +, 0, \odot_i, 1_i, (-)^{*_i})$ is a Kleene algebra for $0 \le i < n$, and for $0 \le i < j < n$, the *Kleene plus* operator $(-)^{+_j}$, defined by $A^{+_j} = A \odot_j A^{*_j}$, is a lax morphism with respect to the i-multiplication of j-dimensional elements on the right (resp. left) in the sense that for all $A \in K$ and $\phi \in K_j$,

$$\phi \odot_i A^{+_j} \le (\phi \odot_i A)^{+_j}, \qquad \text{and} \qquad (\text{resp. } A^{+_j} \odot_i \phi \le (A \odot_i \phi)^{+_j}). \qquad (15)$$

When the underlying semiring is globular and modal, we say that K is a *globular modal n-Kleene algebra*.

To provide a link to the polygraphic case, we remark that the power set of the set of n-cells in the free n-category generated by an n-polygraph P constitutes a globular modal n-Kleene algebra $K(P)$. Following this intuition, an element A of a globular modal n-Kleene algebra will be represented with respect to its i- and j-(co)domains as in the adjacent diagram.

$$\begin{tikzcd}
d_i(A) \arrow[r, bend left, "d_j(A)"] \arrow[r, bend right, "r_j(A)"'] \arrow[r, phantom, "\Downarrow A"] & r_i(A)
\end{tikzcd}$$

4 Church-Rosser theorem

Let K be a globular modal n-Kleene algebra and $0 \le i < j < n$. Given elements ϕ and ψ of K_j, we say that $A \in K$ is a i-*confluence filler* for (ϕ, ψ) if $|A\rangle_j(\psi^{*_i} \odot_i \phi^{*_i}) \ge \phi^{*_i} \odot_i \psi^{*_i}$.

In the modal n-Kleene algebra $K(P)$ corresponding to a polygraph P, taking $i = n-2$, $j = n-1$, letting ψ be the set of $(n-1)$-cells of P and letting ϕ be the set of their inverses, this signifies that A is a set of n-cells containing a filling cell for a confluence diagram corresponding to each branching of $(n-1)$-cells.

4.1. Theorem (Coherent Church-Rosser in globular n-MKA). *Let K be a globular modal n-Kleene algebra. Given $\phi, \psi \in K_j$, for $0 < j < n$, for any i-confluence filler $A \in K$ for (ϕ, ψ), we have*

$$|\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i}) \ge (\phi + \psi)^{*_i},$$

where $\hat{A} := (\phi + \psi)^{*_i} \odot_i A \odot_i (\phi + \psi)^{*_i}$.

Note that the expression $|\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i}) \ge (\phi + \psi)^{*_i}$ signifies that the domain of $\hat{A}^{*_j}$, when restricted on the right to confluences $\psi^{*_i}\phi^{*_i}$, contains at least all of the zig-zags $(\phi + \psi)^{*_i}$.

To finish this section, we provide a sketch of the proof, a detailed version of which can be found in [2]. To ease notation, juxtaposition will denote i-multiplication. By (7) and (15), we have

$$1_i + (\phi + \psi)|\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i}) \le |\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i}) \;\Rightarrow\; (\phi + \psi)^{*_i} \le |\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i})$$

Observing that $1_i \le \psi^{*_i}\phi^{*_i} \le |\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i})$, it remains to show that

$$(\phi + \psi)|\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i}) \le |\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i}).$$

By distributivity, we may prove this for each of the summands. Below, we show the formal calculations in the n-Kleene algebra on the left, which are reflected in the diagrammatic representations on the right:

$$\begin{aligned}
\phi|\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i}) &\le |\phi\hat{A}^{*_j}\rangle_j(\phi\psi^{*_i}\phi^{*_i}) \\
&\le |\phi\hat{A}^{*_j}\rangle_j(|A\rangle_j(\psi^{*_i}\phi^{*_i})\phi^{*_i}) \\
&\le |\phi\hat{A}^{*_j}\rangle_j(|A\phi^{*_i}\rangle_j(\psi^{*_i}\phi^{*_i}\phi^{*_i})) \\
&\le |\phi\hat{A}^{*_j} \odot_j A\phi^{*_i}\rangle_j(\psi^{*_i}\phi^{*_i}) \\
&\le |\hat{A}^{*_j} \odot_j \hat{A}\rangle_j(\psi^{*_i}\phi^{*_i}) \\
&\le |\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i})
\end{aligned}$$

$$\begin{tikzcd}
\cdot \arrow[r, "\phi"] \arrow[ddr, "\psi^{*_i}"'] & \cdot \arrow[rr, "(\phi+\psi)^{*_i}"] \arrow[dr, "\psi^{*_i}"] & & \cdot \\
& \arrow[r, phantom, "\Downarrow A\phi^{*_i}"] & \cdot \arrow[ur, "\phi^{*_i}"'] \arrow[u, phantom, "\Downarrow \phi\hat{A}^{*_j}"] & \\
& \cdot \arrow[ur, "\phi^{*_i}"'] & &
\end{tikzcd}$$

$$\begin{aligned} \psi|\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i}) &\leq |\psi\hat{A}^{*_j}\rangle_j(\psi\psi^{*_i}\phi^{*_i}) \\ &\leq |\psi\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i}) \\ &\leq |\hat{A}^{*_j}\rangle_j(\psi^{*_i}\phi^{*_i}) \end{aligned}$$

$$\begin{tikzcd} \cdot \arrow[r, "\psi"] \arrow[rrd, bend right, "\psi^{*_i}"'] & \cdot \arrow[rr, leftrightarrow, "(\phi+\psi)^{*_i}"] \arrow[rd, "\psi^{*_i}"] \arrow[d, phantom, "="] & & \cdot \\ & {} & \cdot \arrow[ru, "\phi^{*_i}"'] \arrow[u, phantom, "\Downarrow \psi\hat{A}^{*_j}"] & \end{tikzcd}$$

.

5 Conclusion

By introducing the structure of modal n-Kleene algebra, we have provided an algebraic framework for coherent confluence proofs in higher dimensional rewriting theory. A more detailed account of this structure and its properties can be found in [2]. There, in addition to the Church-Rosser theorem, a notion of termination in modal n-Kleene algebras is formalised, and a coherent version of Newman's lemma is formulated and proved in the setting of globular modal n-Kleene algebra.

Perspectives for future work most notably include a formulation of the coherent confluence Squier theorem [5] for abstract rewriting systems in globular modal n-Kleene algebras. This result concerns homotopical properties of polygraphs considered as cofibrant objects in the folk model structure of ω-categories. A central goal is thus to formalise the notion of cofibrant object in the language of modal n-Kleene algebras, and thereby relate these to questions in homotopy type theory. Other avenues of future work include formalising the structure of modal n-Kleene algebra in the proof assistant Isabelle, following [1], and developing a cubical approach.

References

[1] A. Armstrong, V. Gomes, and G. Struth. Algebras for program correctness in Isabelle/HOL. volume 8428, 04 2014.

[2] C. Calk, E. Goubault, P. Malbos, and G. Struth. Algebraic coherent confluence and higher-dimensional Kleene algebras. *Work in progress*, 2020.

[3] J. Desharnais and G. Struth. Internal axioms for domain semirings. *Science of Computer Programming*, 76(3):181–203, 2011.

[4] Y. Guiraud, E. Hoffbeck, and P. Malbos. Convergent presentations and polygraphic resolutions of associative algebras. *Math. Z.*, 293(1-2):113–179, 2019.

[5] Y. Guiraud and P. Malbos. Polygraphs of finite derivation type. *Math. Structures Comput. Sci.*, 28(2):155–201, 2018.

[6] B. Jónsson and A. Tarski. Boolean algebras with operators. Part I. *American Journal of Mathematics*, 73(4):891–939, 1951.

[7] G. Struth. Abstract abstract reduction. *J. Log. Algebr. Program.*, 66(2):239–270, 2006.

Safety and Completeness of Disambiguation corresponds to Termination and Confluence of Reordering

Luís Eduardo de Souza Amorim[1] and Eelco Visser[2]

[1] Australian National University, Australia
LuisEduardo.deSouzaAmorim@anu.edu.au
[2] Delft University of Technology, The Netherlands
e.visser@tudelft.nl

Abstract

Associativity and priority are well known techniques to disambiguate expression grammars. In recent work we develop a direct semantics for disambiguation by associativity and priority rules and prove that a safe and complete disambiguation relation produces a safe and complete disambiguation. The proof approach relies on a correspondence between disambiguation and term rewriting such that safety of disambiguation corresponds to termination of the rewrite system and completeness of disambiguation correspond to confluence of the rewrite system. In this extended abstract we illustrate that approach using diagrams.

1 Introduction

Associativity and priority are well known techniques to disambiguate expression grammars. Figure 1 illustrates the approach. An expression grammar defines the infix operators of an expression language using left- and right-recursive productions such as `Exp.Add = Exp "+" Exp`. Such a grammar is ambiguous; an expression such as `a + b + c` can be read as `(a + b) + c` or as `a + (b + c)`. One way to disambiguate an expression grammar is to transform it to a grammar that uses extra non-terminals to represent priority levels. However such grammars are harder to read and the direct correspondence to the underlying abstract syntax trees is lost. An alternative approach is to augment an ambiguous expression grammar with associativity and priority rules. The semantics of such disambiguation rules is typically defined *indirectly* in the implementation of parser generators or by means of grammar transformations. In recent work, we have developed a *direct semantics* for associativity and priority in terms of subtree exclusion that extends to expression grammars with prefix and postifix operators, mixfix operators, indirect recursion, and overlap. We are currently revising a paper about this work for the TOPLAS journal [2]. A previous version of the semantics of disambiguation rules appeared in [1], but did not feature the proof technique based on rewriting. We refer to those papers for a discussion of related work.

```
lexical syntax
  ID = [a-zA-Z][a-zA-Z0-9]*
context-free syntax
  Exp.Var = ID
  Exp.Add = Exp "+" Exp {left}
  Exp.Min = Exp "-" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Div = Exp "/" Exp {left}
  Exp.Pow = Exp "^" Exp {right}
context-free priorities
  Exp.Pow
  > {left: Exp.Mul Exp.Div}
  > {left: Exp.Add Exp.Min}
```

Figure 1: SDF3 definition.

To verify the approach we developed a technique based on term rewriting that shows that soundness and completeness of disambiguation corresponds to termination and confluence reordering parse trees. In this extended abstract we illustrate the proof technique for the case of infix expression grammars. We omit formal definitions and mostly explain the approach using diagrams.

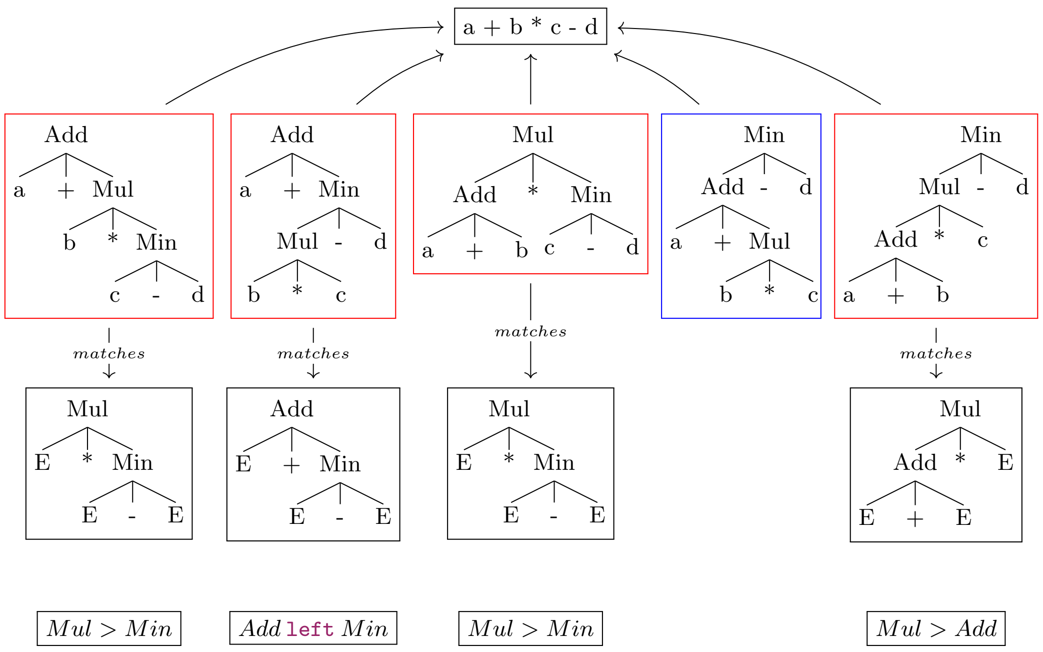
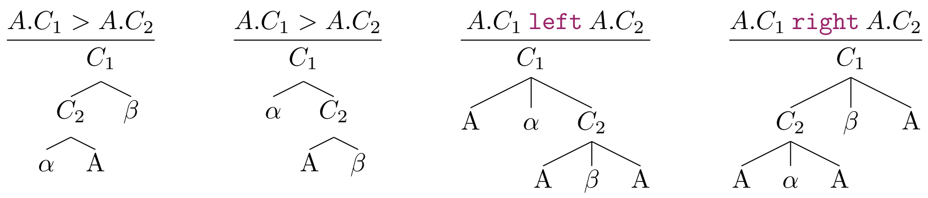


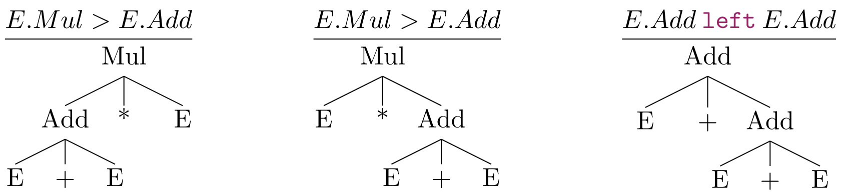
Figure 2: Parse trees for an ambiguous expression, and its disambiguation using subtree exclusion rules

2 Disambiguation by Subtree Exclusion

An ambiguous sentence has multiple parse trees. The diagram in figure 2 shows the parse trees for the expression `a + b * c - d` according the underlying grammar of figure 1. Disambiguation by subtree exclusion defines conflict patterns that should not occur in selected parse trees. The safe subtree exclusion rules (for infix expressions) of SDF3 [2] are defined as follows:



Instantiating these rules to some of the disambiguation rules in figure 1 leads to the following subtree exclusion patterns (aka conflict patterns):

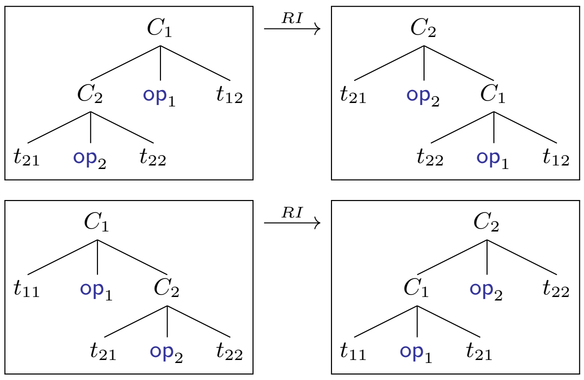


Applying these rules to the expression in figure 2 shows that the sentence is completely disambiguated as all but one parse tree for the expression are rejected.

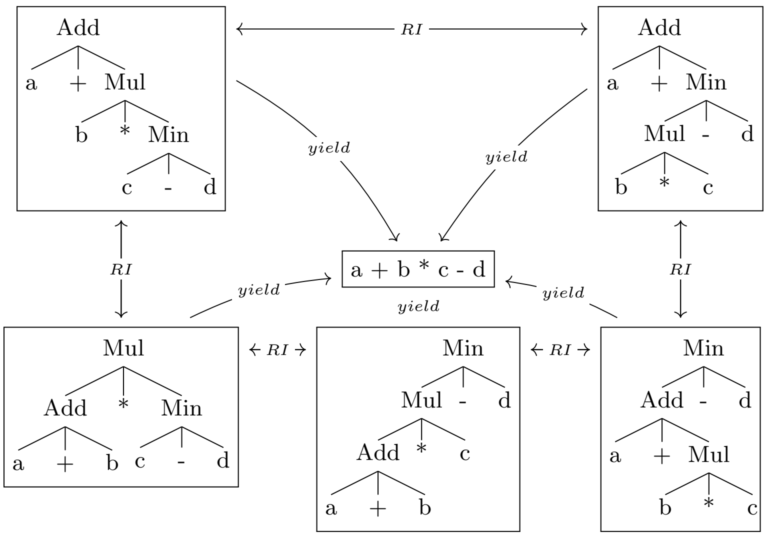
A set of disambiguation rules is *safe* when each sentence in the underlying grammar is also a sentence in the disambiguated grammar. That is, no sentences are excluded. A set of disambiguation rules is *complete* when each sentence in the underlying grammar has at most one parse tree in the disambiguated grammar. That is, each sentence is completely disambiguated. How can we prove that set of disambiguation rules safe and complete? That is a central question in our work on the semantics disambiguation rules [2].

3 Proving Safety and Completeness

The central insight in our approach to proving safety and completeness of disambiguation is that disambiguation by means of associativity and priority rules corresponds to reordering of parse trees. For infix expression grammars we define a rewrite system generated by instiantiating the following rewrite rule schemas for each pair of productions in a grammar:


C_1
C_2 op_1 t_{12}
t_{21} op_2 t_{22}
RI
C_2
t_{21} op_2 C_1
t_{22} op_1 t_{12}
C_1
t_{11} op_1 C_2
t_{21} op_2 t_{22}
RI
C_2
C_1 op_2 t_{22}
t_{11} op_1 t_{21}

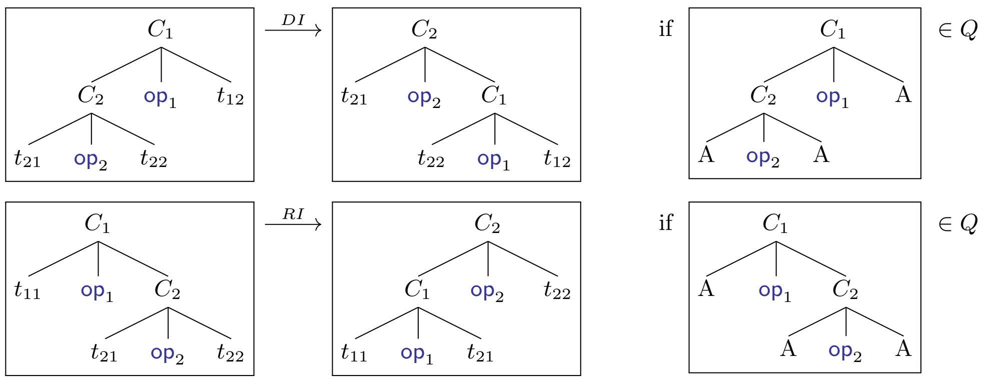
When we apply these rules to the expression in figure 2, we see that all trees can be converted into each other using these rewrite rules:


Add a + Mul b * Min c - d
RI
Add a + Min Mul b * c - d
yield
a + b * c - d
Mul Add a + b * Min c - d
Min Mul Add a + b * c - d
Min Add a + Mul b * c - d

In general, we have a theorem that states that all ambiguities in an infix expression grammar are related by reordering, expressed diagrammatically as follows:

$$\begin{tikzcd}
 & & w & & \\
t_1 \arrow[urr, bend left, "\mathsf{yield}"] \arrow[r, leftrightarrow, "\mathsf{RI}"] & \dots \arrow[r, leftrightarrow, "\mathsf{RI}"] & t_i \arrow[u, "\mathsf{yield}"] \arrow[r, leftrightarrow, "\mathsf{RI}"] & \dots \arrow[r, leftrightarrow, "\mathsf{RI}"] & t_n \arrow[ull, bend right, "\mathsf{yield}"']
\end{tikzcd}$$

The reordering rewrite system is non-terminating. Each tree can be converted in each other tree. Using the subtree exclusion patterns generated from the associativity and priority rules of a grammar, we direct the rules of the rewrite system, leading to the following rule schemas:



In our paper [2] we show that safety of disambiguation corresponds to termatination of this rewrite system and that completeness of disambiguation corresponds to confluence of the rewrite system. We illustrate that here using the diagrams in figures 3, 4, and 5.

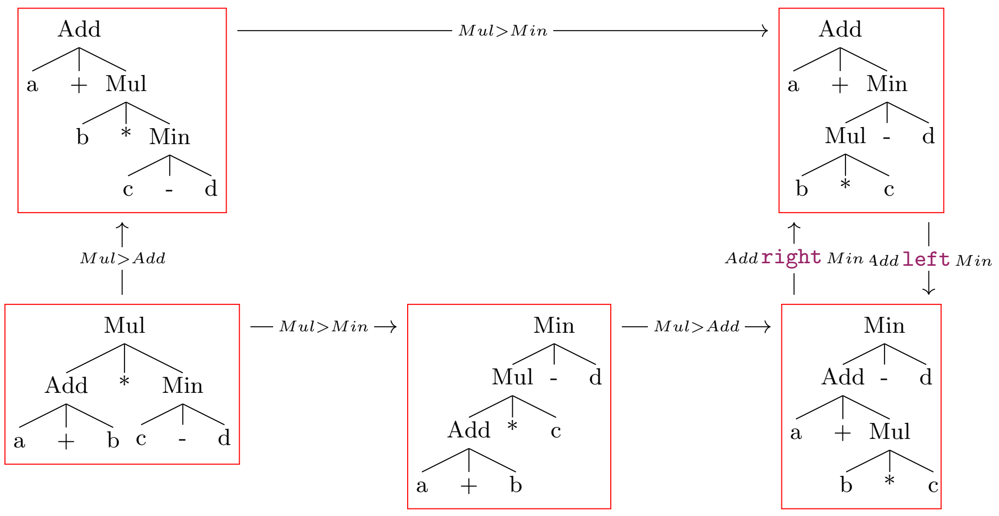


Figure 3: Unsafety corresponds to non-termination

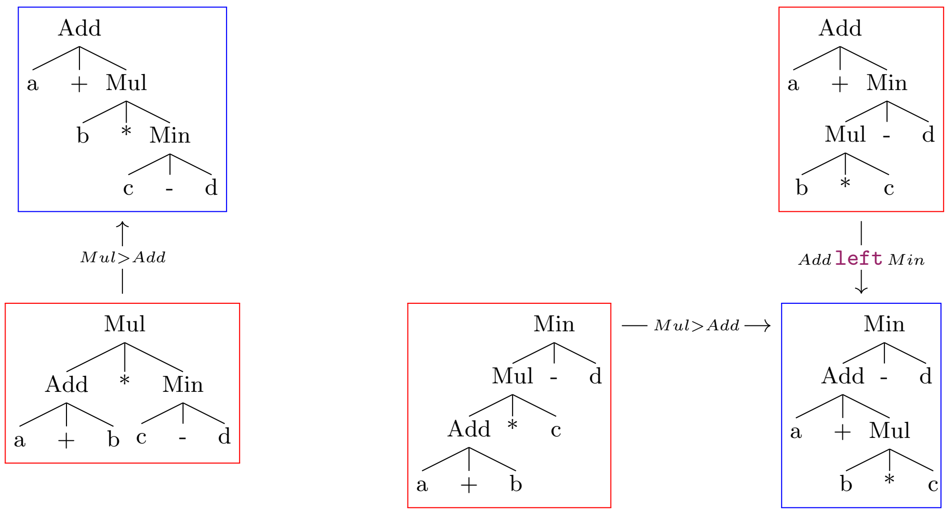


Figure 4: Incompleteness corresponds to non-confluence (non-Church Rosser)

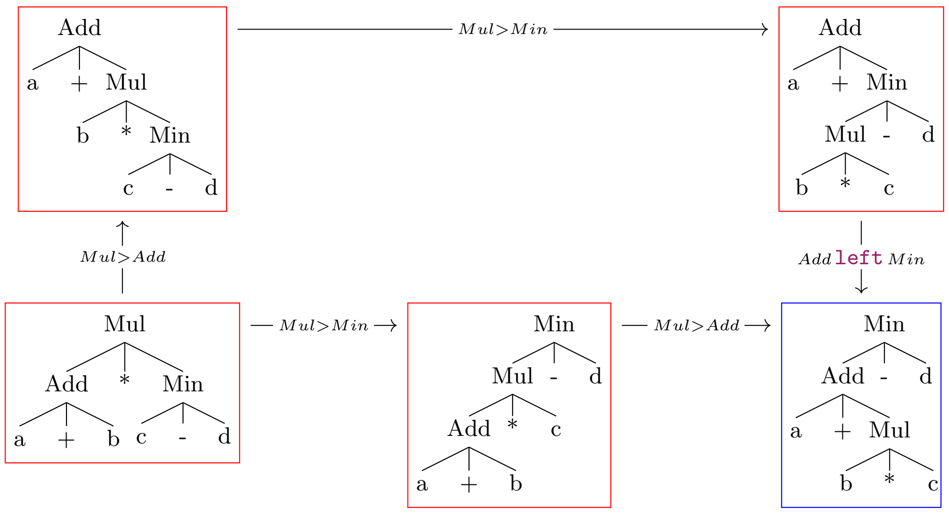


Figure 5: Safety and completeness correspond to termination and confluence.

References

[1] Luís Eduardo de Souza Amorim. *Declarative Syntax Definition for Modern Language Workbenches*. PhD thesis, Delft University of Technology, 2019.

[2] Luís Eduardo de Souza Amorim and Eelco Visser. A direct semantics for declarative disambiguation of expression grammars. *ACM Transactions on Programming Languages*, 2020. under revision.

Confluence of drag rewriting

Jean-Pierre Jouannaud[1] **and Fernando Orejas**[2]

1 **École Normale de Paris-Saclay, Laboratoire de Spécification et Vérification, France**
2 **Universitat Politècnica de Catalunya, Barcelona, Spain**

Abstract

We develop a confluence result for graph rewriting based on the drag model [5].

1 Introduction

Rewriting with graphs has a long history in computer science, graphs being used to represent data structures, but also program structures and even concurrent and distributed computational models. They therefore play a key rôle in program evaluation, transformation, and optimization, and more generally program analysis; see, for example, [2].

Our work is based on a recent, purely combinatorial view of graphs [5]. To assess our claim that drags are a natural generalization of terms, we extend the most useful term rewriting techniques to drags: the recursive path ordering [4]; unification [8]; confluence in this paper. Our main result is that confluence of a terminating set of drag rewrite rules can be decided by the usual joinability test of their critical pairs. Comparisons with the literature will be done at length in a forthcoming paper.

2 The drag model [5]

Drags are finite, ***d***irected, ordered, ***r***ooted, l***a***beled multi-***g***raphs. Vertices with no outgoing edges are designated *sprouts*. Other vertices are *internal*. We presuppose: a set of function symbols Σ, whose elements f, equipped with a fixed arity $|f|$, are used as labels for internal vertices; and a set of nullary variable symbols Ξ, disjoint from Σ, used to label sprouts.

▸ **Definition 1** (Drags)**.** A *drag* is a tuple $\langle V, R, L, X, S\rangle$, where

1. V is a finite set of *vertices*, denoted by $\mathcal{V}er(D)$;
2. $R : [p \,..\, p + |R|] \to V$ is a finite list of vertices, called *roots*; $R(p + n - 1)$ refers to the nth root in R; unless otherwise stated, $p = 1$; we use $\mathcal{R}(D)$ for the roots of D;
3. $S \subseteq V$ is a set of *sprouts*, leaving $V \setminus S$ to be the *internal* vertices;
4. $L : V \to \Sigma \cup \Xi$ is the *labeling* function, mapping internal vertices $V \setminus S$ to labels from the vocabulary Σ and sprouts S to labels from the vocabulary Ξ, writing $v : f$ for $f = L(v)$;
5. $X : V \to V^*$ is the *successor* function, mapping each vertex $v \in V$ to a list of vertices in V whose length equals the arity of its label (that is, $|X(v)| = |L(v)|$).

The reflexive-transitive closure X^* of the relation X is called *accessibility*. Vertex v is *accessible* if it is accessible from some root. A drag is *clean* if all its vertices are accessible. Terms as ordered trees, sequences of terms (forests), terms with shared subterms (dags) and sequences of dags (jungles) are all particular kinds of clean rooted drags.

Sprouts may be roots, this is essential for having a nice algebra of drags.

We use $\mathcal{V}ar(D)$ for the set of variables labeling the sprouts of D. A drag is *linear* if no two sprouts have the same label, in which case variables and sprouts can be identified.

2 Confluence of drag rewriting

2.1 Drag composition

A variable in a drag should be understood as a potential connection to a root of another drag, as specified by a connection device called a *switchboard*. A switchboard ξ is a pair of partial injective functions, one for each drag, whose *domain* $\mathcal{D}om(\xi)$ and *image* $\mathcal{I}m(\xi)$ are a set of sprouts of one drag and a set of positions in the list of roots of the other, respectively.

▸ **Definition 2** (Switchboard)**.** Let $D = \langle V, R, L, X, S\rangle$ and $D' = \langle V', R', L', X', S'\rangle$ be drags. A *switchboard* ξ for D, D' is a pair $\langle \xi_D : S \to [1..|R'|]; \xi_{D'} : S' \to [1..|R|]\rangle$ of partial injective functions such that

1. $s \in \mathcal{D}om(\xi_D)$ and $L(s) = L(t)$ imply $t \in \mathcal{D}om(\xi_D)$ and $\xi_D(s) = \xi_D(t)$ for all sprouts $s, t \in S$;
2. $s \in \mathcal{D}om(\xi_{D'})$ and $L'(s) = L'(t)$ imply $t \in \mathcal{D}om(\xi_{D'})$ and $\xi_{D'}(s) = \xi_{D'}(t)$ for all $s, t \in S'$;
3. ξ is *well-behaved*: it does not induce any cycle among sprouts, using ξ, R, R' relationally:
$$\nexists\ n > 0, s_1, \dots, s_{n+1} \in S, t_1, \dots, t_n \in S', s_1 = s_{n+1}.\ \forall i \in [1..n].\ s_i\ \xi_D R' X'^*\ t_i\ \xi_{D'} R X^*\ s_{i+1}$$

The pair $\langle D', \xi\rangle$ is an *extension* of D, a *rewriting extension* if ξ_D is surjective and $\xi_{D'}$ total.

Sprouts labelled by the same variable should be connected to the same vertex, as stipulated by conditions (1,2). It follows that $\xi_D(\mathcal{D}om(\xi_D))$ must be a set, making the set difference $[1..|R'|] \smallsetminus \xi_D(\mathcal{D}om(\xi_D))$ well defined.

We now move to the composition operation on drags induced by a switchboard. The essence of this operation is that the (disjoint) union of the two drags is formed, but with sprouts in the domain of the switchboards merged with the roots to which the switchboard images refer. Merging sprouts with their images requires one to worry about the case where multiple sprouts are merged successively, when the switchboards map sprout to rooted-sprout to rooted-sprout, until, eventually, an internal vertex of one of the two drags must be reached because a switchboard is well-behaved. That vertex is called *target*:

▸ **Definition 3** (Target)**.** Let $D = \langle V, R, L, X, S\rangle$ and $D' = \langle V', R', L', X', S'\rangle$ be drags such that $V \cap V' = \varnothing$, and ξ be a switchboard for D, D'. The *target* $\xi^*(s)$ is a mapping from sprouts in $S \cup S'$ to vertices in $V \cup V'$ defined as follows:

Let $v = R'(n)$ if $s \in S$, and $v = R(n)$ if $s \in S'$, where $n = \xi(s)$.

1. If $v \in (V \cup V') \smallsetminus (S \cup S')$, then $\xi^*(s) = v$.
2. If $v \in (S \cup S') \smallsetminus \mathcal{D}om(\xi)$, then $\xi^*(s) = v$.
3. If $v \in \mathcal{D}om(\xi)$, then $\xi^*(s) = \xi^*(v)$.

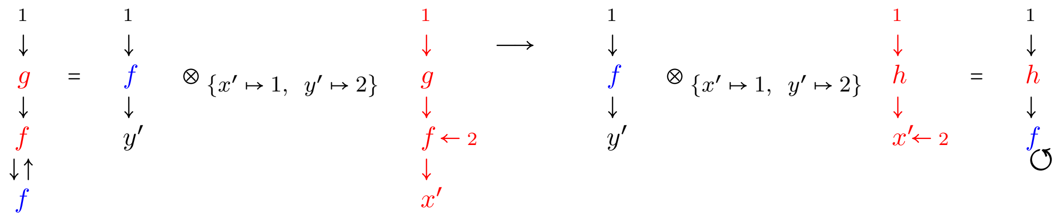
The target mapping $\xi^*(\_)$ is extended to all vertices of D and D' by letting $\xi^*(v) = v$ when $v \in (V \smallsetminus S) \cup (V' \smallsetminus S')$.

We are now ready for defining the composition of two drags. Its set of vertices will be the union of two components: the internal vertices of both drags, and their sprouts which are not in the domain of the switchboard. The labeling is inherited from that of the components.

▸ **Definition 4** (Composition)**.** Let $D = \langle V, R, L, X, S\rangle$ and $D' = \langle V', R', L', X', S'\rangle$ be drags such that $V \cap V' = \varnothing$, and let ξ be a switchboard for D, D'. Their *composition* is the drag $D \otimes_\xi D' = \langle V'', R'', L'', X'', S''\rangle$, with interface (R'', S'') denoted $(R, S) \otimes_\xi (R', S')$, where

1. $V'' = (V \cup V') \smallsetminus \mathcal{D}om(\xi)$;
2. $S'' = (S \cup S') \smallsetminus \mathcal{D}om(\xi)$;
3. $R'' = \xi^*(R([1..|R|] \smallsetminus \xi_{D'}(\mathcal{D}om(\xi_{D'})))) \cup \xi^*(R'([1..|R'|] \smallsetminus \xi_D(\mathcal{D}om(\xi_D))))$;
4. $L''(v) = L(v)$ if $v \in V \cap V''$; and $L''(v) = L'(v)$ if $v \in V' \cap V''$;
5. $X''(v) = \xi^*(X(v))$ if $v \in V \smallsetminus S$; and $X''(v) = \xi^*(X'(v))$ if $v \in V' \smallsetminus S'$

If $\langle D, \xi\rangle$ is a rewriting extension of D', then *all* roots and sprouts of D' disappear in the composed drag. The drag D can then be seen as the context of the left-hand side of a rule $D' \to R$, where R must have the same number of roots as D' (and $\mathcal{V}ar(R) \subseteq \mathcal{V}ar(D')$.)



$=$
$\otimes \{x' \mapsto 1,\ y' \mapsto 2\}$
$\longrightarrow$
$\otimes \{x' \mapsto 1,\ y' \mapsto 2\}$
$=$

Figure 1 Rewriting and cycles.

2.2 Drag rewriting

Rewriting with drags is similar to rewriting with trees: we first select an instance of the left-hand side L of a rule in a drag D by exhibiting an extension $\langle W, \xi \rangle$ such that $D = W \otimes_\xi L$ – this is drag matching, then replace L by the corresponding right-hand side R in the composition. A very important condition for the result to be a drag is, accordingly, that the left- and right-hand sides of rules have the same number of roots:

▸ **Definition 5** (Rules). A *drag rewrite rule* is a pair of clean drags, written $\mathrm{L} \to \mathrm{R}$, such that (i) $|\mathcal{R}(\mathrm{L})| = |\mathcal{R}(\mathrm{R})|$, and (ii) $\mathcal{V}ar(\mathrm{R}) \subseteq \mathcal{V}ar(\mathrm{L})$. A drag rewriting system is a set of drag rewrite rules.

Condition (i) ensures that L and R have a perfect fit with any same environment, that is, both can be composed with any extension of L. Condition (ii) is standard for rewrite rules.

▸ **Definition 6** (Rewriting). Let $\mathcal{S}$ be a drag rewrite system. We say that a nonempty clean drag D *rewrites* to a clean drag D', and write $D \longrightarrow_{\mathcal{S}} D'$, iff $D = C \otimes_\xi \mathrm{L}$ and $D' = C \otimes_\xi \mathrm{R}$ for some drag rewrite rule $\mathrm{L} \to \mathrm{R} \in \mathcal{S}$ and clean rewriting extension $\langle C, \xi \rangle$ of L.

Because ξ is a rewriting switchboard, ξ_C must be linear, implying that the variables labeling the sprouts of C that are not already sprouts of D must all be different. Then, ξ_C must be surjective, implying that the roots of L (hence those of R) disappear in the composition, a case where the composition is commutative –we shall mostly write the context on the left, though. Further, ξ_{L} must be total, implying that the sprouts of L (hence those of R) disappear in the composition. Finally, D and C being clean, it is easy to show that D' is clean as well, which is therefore a property rather than a requirement.

▸ **Example 7** (Figure 1). The (red) rewrite rule $g(f(x')) \to h(x')$, whose roots are g and f on the left-hand side and h and x' on the right-hand side, applies with a blue context, colours which are reflected in the input and output terms (the rule applies across the cycle).

We are now finished with the material from [5] needed for the rest of this paper.

2.3 Drag unification [8]

The purpose here is to *identify* two clean drags U, V by composing them with the same *minimal* rewriting context $\langle C, \xi \rangle$, resulting in the same drag W. An identification corresponds to the fact that we want the same drag to be rewritten by two different rewrite rules whose left-hand sides are U and V. In order for $C \otimes_\xi U$ and $C \otimes_\xi V$ to both make sense, we assume that U, V are renamed apart (variables and root numbers).

▸ **Definition 8.** Given drags U, V, we call *partner vertices* two lists L_U, L_V of equal length of internal vertices of U and V, respectively, such that no two vertices $u, u' \in L_U$ (resp., $v, v' \in L_V$) are in relationship with X_U (resp., X_V).

4 Confluence of drag rewriting

▸ **Definition 9.** Two drags U, V are *identified* with a drag W at *partner vertices* $(\overline{u}, \overline{v})$ by an injective function $o : \mathcal{V}er(U) \cup \mathcal{V}er(V) \to \mathcal{V}er(W)$ called *identification*, written $U[\overline{u}] =_o V[\overline{v}]$, iff:

1. $o(\overline{u}) = o(\overline{v})$;
2. $\forall w \in \mathcal{V}er(U), w' \in \mathcal{V}er(V)$ such that $o(w) = o(w')$, $w : f$ iff $w' : f$ iff $o(w) = o(w') : f$;
3. $\forall w \in \mathcal{V}er(U), w' \in \mathcal{V}er(V)$ such that $o(w) = o(w')$, $o(X_U(w)) = o(X_V(w))$.

While two terms u, v are unified *at their root*, the solution being a substitution σ such that $u\sigma = v\sigma$, two drags U, V are unified at partner vertices $(\overline{u}, \overline{v})$, the solution being an extension $\langle C, \xi \rangle$ of both U and V that identifies $C \otimes_\xi U$ and $C \otimes_\xi V$ at these partner vertices:

▸ **Definition 10.** A *unification problem* is a pair of clean drags (U, V) that are renamed apart, together with partner vertices $P = \{(\overline{u}, \overline{v})\}$, which we write $U[\overline{u}] = V[\overline{v}]$. A *solution* (or *unifier*) to the unification problem $U[\overline{u}] = V[\overline{v}]$ is a clean rewriting extension $\langle C, \xi \rangle$ such that the *overlap* drags $C \otimes_\xi U$ and $C \otimes_\xi V$ are identified at P. A unification problem $U[\overline{u}] = V[\overline{v}]$ is *solvable* if it has a solution.

We want unification to be minimal, that is, to capture all possible extensions that identify U and V, without useless identifications occuring above or below partner vertices.

▸ **Definition 11.** We say that a drag U is an *instance* of a drag V, or that V *subsumes* U, and write $U \geq V$, if there exists a clean context extension $\langle C, \xi \rangle$ such that $U = C \otimes_\xi V$.

Cleanness is essential here, since otherwise any drag would be an instance of any other drag, which is also the reason why rewriting considers clean extensions only. In the following, we assume for convenience that the sprouts of U, V are labelled by different sets of variables.

▸ **Lemma 12.** *$\geq$ is a quasi-order whose equivalence is variable renaming and strict part is a well-founded order.*

We now extend the (of course well-founded) subsumption order to context extensions:

▸ **Definition 13.** Let U be a drag, of which $\langle C, \xi \rangle$ and $\langle D, \zeta \rangle$ are two context extensions. We say that (D, ζ) is an *instance* of (C, ξ) (or that (C, ξ) *subsumes* (D, ζ)) w.r.t. U, and write $(D, \zeta) \geq_U (C, \xi)$, if $(D \otimes_\zeta U)$ is an instance of $(C \otimes_\xi U)$.

We show in [8] the following key result:

▸ **Theorem 14.** *Given a unification problem, there is a unique most general unifying extension, if any, computable in quadratic time.*

3 Confluence

Confluence of a terminating term rewriting system follows from the joinability of its critical pairs, obtained by unifying overlapping left-hand sides of rules [9]. Our goal is to generalize this result to the drag framework. For non-terminating countable systems, replacing joinability by the existence of decreasing diagrams [10] should also work.

▸ **Lemma 15.** *Let $S \longleftarrow_{\mathrm{L}\to\mathrm{R}} U \longrightarrow_{\mathrm{G}\to\mathrm{D}} T$, and assume that U has no internal vertex being at the same time an internal vertex of* L *and of* G*. Then, there exist two drags V, W and a switchboard ξ such that $U = V \otimes_\xi W$, $V \longrightarrow_{\mathrm{L}\to\mathrm{R}} V'$, $W \longrightarrow_{\mathrm{G}\to\mathrm{D}} W'$, $S = V' \otimes_\xi W$ and $T = V \otimes_\xi W'$.*

Proof. By definition of rewriting, there exist rewriting extensions $\langle A, \xi \rangle$ and $\langle B, \zeta \rangle$ such that $U = A \otimes_\xi \mathrm{L} = B \otimes_\zeta \mathrm{G}$. We assume without loss of generality that L and G are renamed apart as well as A and B, making $\xi_\mathrm{L} \cup \zeta_\mathrm{G}$ well defined, as well $\mathcal{R}(A) \cup \mathcal{R}(B)$.

Let now C be the drag whose internal vertices are those of U which are not internal vertices of either $\textsc{l}$ or $\textsc{g}$, its roots and sprouts are those of A plus those of B, and its successor relationship is inherited from that of U.

Since $\textsc{l}$ and $\textsc{g}$ do not share internal vertices by assumption, $B = \textsc{l} \otimes_\xi C$ while $A = \textsc{g} \otimes_\zeta C$, hence $U = \textsc{l} \otimes_\xi C \otimes_\zeta \textsc{g}$ (using associativity of composition [5]). We then take $V = \textsc{l}$ and $W = C \otimes_\zeta \textsc{g}$. ◂

▸ **Lemma 16** (Commutation). *Assume that $U = V \otimes_\xi W$, $V \longrightarrow_{\textsc{l}\to\textsc{r}} V'$ and $W \longrightarrow_{\textsc{g}\to\textsc{d}} W'$. Then, $U \longrightarrow_{\textsc{l}\to\textsc{r}} V' \otimes_\xi W \longrightarrow_{\textsc{g}\to\textsc{d}} V' \otimes_\xi W'$ and $U \longrightarrow_{\textsc{g}\to\textsc{d}} V \otimes_\xi W' \longrightarrow_{\textsc{l}\to\textsc{r}} V' \otimes_\xi W'$.*

Proof. Easy consequence of associativity of composition. ◂

▸ **Lemma 17** (Critical overlap). *Let $S \longleftarrow_{\textsc{l}\to\textsc{r}} U \longrightarrow_{\textsc{g}\to\textsc{d}} T$, and let us assume that U has an internal vertex w which is an internal vertex of both $\textsc{l}$ and $\textsc{g}$. Then, there exist $\overline{u} \in \mathcal{V}er(\textsc{l})$ and $\overline{v} \in \mathcal{V}er(\textsc{g})$, and a unifying extension $\langle E, \zeta \rangle$ of the equation $\textsc{l}[\overline{u}] = \textsc{g}[\overline{v}]$ such that $U = \textsc{l} \otimes_\zeta E = \textsc{g} \otimes_\zeta E$.*

Proof. Let A be the subset of internal vertices of U which are also internal vertices of $\textsc{l}$ and of $\textsc{g}$, roots of $\textsc{l}, U, \textsc{g}$ being considered as specific internal vertices. By assumption, $A \neq \emptyset$, implying that $\textsc{l}$ and $\textsc{g}$ overlap. The core of the proof is the definition of two lists $\overline{v}, \overline{w}$ of partner vertices of $\textsc{l}, \textsc{g}$ which generate A, that is, all vertices of A are accessible from $\overline{v}$ in $\textsc{l}$ and from $\overline{w}$ in $\textsc{g}$. As partner vertices, v_i and w_i must coincide, that is, be identified in A. Let us denote vertices of A by u, v and w according to their origin, in $U, \textsc{l}$ and $\textsc{g}$ respectively.

Because left-hand sides of rules are clean drags, for all internal vertices $u \in A$, there exists some root $r \in \mathcal{R}(\textsc{l}) \cup \mathcal{R}(\textsc{g})$ such that, $r\,(X^*_{\textsc{l}} \cup X^*_{\textsc{g}})\, v$. There are therefore three kinds of partner vertices (v, w): v, w are roots of $\textsc{l}, \textsc{g}$; or v is a root of $\textsc{l}$ and w is not a root of $\textsc{g}$; or v is not a root of $\textsc{l}$ and w is a root of $\textsc{g}$. Eliminating redundancies (with respect to accessibility) yields two lists of vertices that satisfy the conditions for being partner vertices, and generate A.

We now show that U defines a unifying extension of the equation $\textsc{l}[\overline{v}] = \textsc{g}[\overline{w}]$. Since $\textsc{l}$ and $\textsc{g}$ match U, $U = \textsc{l}[\overline{v}] \otimes_\gamma C = \textsc{g}[\overline{w}] \otimes_\delta D$ for some rewriting extensions $\langle C, \gamma \rangle$ and $\langle D, \delta \rangle$ of $\textsc{l}$ and $\textsc{g}$ respectively. Assuming that $\textsc{l}, \textsc{g}$ are renamed apart, then $U = (\textsc{l}[\overline{v}] \oplus \textsc{g}[\overline{w}]) \otimes_{\gamma \cup \delta} (C \oplus D)$. Hence $\textsc{l}$ and $\textsc{g}$ are unifiable at partner vertices $(\overline{v}, \overline{w})$ with the extension $(C \oplus D, \gamma \cup \delta)$. ◂

▸ **Definition 18** (Critical pair). Let $\textsc{l} \to \textsc{r}$ and $\textsc{g} \to \textsc{d}$ be two rules that are unifiable at partner vertices $\overline{v}, \overline{w}$, and $\langle C, \xi \rangle$ be an mgu. Then, $\langle \textsc{l} \otimes_\xi C, \textsc{g} \otimes_\xi C \rangle$ is called a *critical pair* of $\textsc{l}, \textsc{g} \to \textsc{d}$ at $\overline{v}, \overline{w}$.

Note that the notion of drag critical pair becomes symmetric.

Given a drag rewriting system, how many critical pairs can be generated ? Their number is indeed bounded by the potential choices for partner vertices, which must satisfy the constraints stated in the proof of Lemma 17. In practice, this number should remain small, as is the case for terms.

We can now end up with our main result, which follows easily from Lemmas 15, 16 and 17:

▸ **Theorem 19.** *Let $\mathcal{S}$ be a terminating rewrite system on drags. Then, $\mathcal{S}$ is confluent iff all its critical pairs are joinable.*

As for terms: (i) termination is not essential: the same result could be rephrased, we believe, by using decreasing diagrams instead of joinability; and (ii) redundancy criteria [1], should allow to filter out useless critical pairs.

4 Conclusion

Drags appear to be an extremely handy generalization of terms, dags and jungles: the intuitions behind them all are very similar, as well as the most important algorithms for implementing rewriting and testing its termination and confluence, despite the possibility of having arbitrary cycles in drags. This is made possible by a powerful composition operator.

Drags do not exactly generalize terms, though, as is pointed out in [5]. This is because our definition of composition *forces* sharing, as does term rewriting in practice. Capturing the term case requires using a composition operator based on drag isomorphism instead of drag equality in presence of non-linear variables in rules. This is of course possible, and is currently being investigated.

Finally, the present result together with the drag path ordering given in [4] shoud allow the development of completion procedures for drag rewriting systems.

Warm thanks to Anne Yenan and José Motos who provided a *deluxe roof* to the first author during his one month stay in Barcelona at the invitation of the second author.

References

1 Leo Bachmair and Harald Ganzinger. On Restrictions of Ordered Paramodulation with Simplification. In Mark E. Stickel editor, *10th International Conference on Automated Deduction*, Kaiserslautern, July 24-27, 1990. Lecture Notes in Computer Science 449, pages 427–441, Springer, 1990.

2 Horatiu Cirstea and David Sabel, editors. *Proceedings Fourth International Workshop on Rewriting Techniques for Program Transformations and Evaluation, WPTE@FSCD 2017, Oxford, UK, September 2017*, volume 265 of *EPTCS*, 2018.

3 Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 243–320. Elsevier, 1990.

4 Nachum Dershowitz and Jean-Pierre Jouannaud. Graph path orderings. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 57 of *EPiC Series in Computing*, pages 307–325. EasyChair, 2018.

5 Nachum Dershowitz and Jean-Pierre Jouannaud. Drags: A compositional algebraic framework for graph rewriting. *Theor. Comput. Sci.*, 777:204–231, 2019.

6 Annegret Habel, Hans-Jörg Kreowski, and Detlef Plump. Jungle evaluation. *Fundam. Inform.*, 15(1):37–60, 1991.

7 Gérard Huet. *Unification dans les langages d'ordre 1, $\ldots, \omega$.* PhD thesis, Université Paris 7, Paris, France, 1976.

8 Jean-Pierre Jouannaud and Fernando Orejas. Unification of drags. In *UNIF 2020.* available at https://hal.inria.fr/hal-02562152.

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

10 Vincent van Oostrom. Confluence by decreasing diagrams. *Theor. Comput. Sci.*, 126(2):259–280, 1994.

Confluence Competition 2020

Aart Middeldorp[1], Naoki Nishida[2], Kiraku Shintani[3], and Johannes Waldmann[4]

[1] Department of Computer Science, University of Innsbruck, Austria
[2] Department of Computing and Software Systems, Nagoya University, Japan
[3] School of Information Science, JAIST, Japan
[4] HTWK Leipzig, Germany

The next few pages in these proceedings contain the descriptions of the tools participating in the 9th Confluence Competition (CoCo 2020). CoCo is a yearly competition in which software tools attempt to automatically (dis)prove confluence and related properties of rewrite systems in a variety of formats. For a detailed description we refer to [1]. This year there were 14 tools (listed in order of registration) participating in 12 categories (listed in order of first appearance in CoCo):

	TRS	CPF-TRS	CTRS	HRS	GCR	CPF-CTRS	UNR	UNC	NFP	COM	INF	SRS
infChecker											✓	
CoLL-Saigawa	✓											✓
Moca											✓	
CSI	✓	✓					✓	✓	✓			✓
FORT-h					✓		✓	✓	✓	✓		
ConCon			✓			✓					✓	
CO3			✓								✓	
CoLL										✓		
nonreach											✓	
ACP	✓	✓	✓				✓			✓		✓
AGCP					✓							
CSI^ho				✓								
CeTA		✓				✓						
SOL				✓								

New this year was that the winning tools[1] of CoCo 2019 participated as demonstration tools, to provide a benchmark to measure progress. Also new is that there are separate winners for YES answers, NO answers, and combined YES/NO answers. The live run CoCo 2020 on StarExec [2] can be viewed at `http://cocograph.uibk.ac.at/2020.html`. Further information about CoCo 2020, including a description of the categories and detailed results, can be obtained from

`http://project-coco.uibk.ac.at/2020/`

References

[1] A. Middeldorp, J. Nagele, and K. Shintani. Confluence Competition 2019. In *Proc. 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 11429 of *LNCS*, pages 25–40, 2019. doi: 10.1007/978-3-030-17502-3_2.

[2] 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 *LNCS (LNAI)*, pages 367–373, 2014. doi: 10.1007/978-3-319-08587-6_28.

[1]They are not listed in the table but see `http://project-coco.uibk.ac.at/2019/results.php`.

infChecker at the 2020 Confluence Competition[*]

Raúl Gutiérrez[1] and Salvador Lucas[2]

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

1 Overview

infChecker 1.0 is a tool for checking *(in)feasibility* of goals $\mathcal{G} = \{F_i\}_{i=1}^m$ where $F_i = (s_{ij} \bowtie_{ij} t_{ij})_{i=1}^{n_i}$ and $\bowtie_{ij} \in \{\to, \to^*, \to^+, \hookrightarrow, \hookrightarrow^*, \hookrightarrow^+, \rhd, \unrhd, \downarrow, \downarrow\!\!\!\hookleftarrow, \leftrightarrow, \hookleftarrow\!\!\!\hookrightarrow, \leftrightarrow^*, \hookleftarrow\!\!\!\hookrightarrow^*\}$ where predicates $\bowtie_{ij}$ represent binary relations on terms (most of them well-known or easy generalizations of well-known relations) defined by provability of goals $s \bowtie_{ij} t$ with respect to a *first-order theories* $\mathsf{Th}_{\bowtie_{ij}}$ [2, 4]. The tool is available here: http://zenon.dsic.upv.es/infChecker/. It is written in Haskell and provides a first implementation of the *Feasibility Framework* [2], where three *processors* have been implemented:

- $\mathsf{P}^{\mathsf{Sat}}$ integrates the satisfiability approach described in [3] to prove infeasibility. In infChecker, we use the model generators AGES [1] and Mace4 [6] to find a proof.
- $\mathsf{P}^{\mathsf{Prov}}$ integrates the logic-based approach to program analysis described in [3] to prove feasibility by theorem proving. In infChecker, we use the theorem prover Prover9 [6].
- P^{NC} adapt the processor that narrow conditions in the 2D DP framework for proving operational termination of CTRs [5] to be used with feasibility sequences.

Our proof strategy is: (1) first, we try to prove feasibility using $\mathsf{P}^{\mathsf{Prov}}$; (2) if $\mathsf{P}^{\mathsf{Prov}}$ fails, we apply $\mathsf{P}^{\mathsf{Sat}}$; (3) if $\mathsf{P}^{\mathsf{Sat}}$ fails, we apply P^{NC}; (4) if P^{NC} succeeds and modifies the feasibility sequence, we go to (2), otherwise we return MAYBE.

References

[1] R. Gutiérrez and S. Lucas. Automatic Generation of Logical Models with AGES. In *CADE 2019: Automated Deduction - CADE 27*, LNCS 11716:287:299. Springer, 2019.

[2] R. Gutiérrez and S. Lucas. Automatically Proving and Disproving Feasibility Conditions. In *Proc. of IJCAR'2020*, LNCS to appear. Springer, 2020.

[3] S. Lucas. Proving semantic properties as first-order satisfiability. *Artificial Intelligence* 277, paper 103174, 24 pages, 2019.

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

[5] S. Lucas, J. Meseguer, and R. Gutiérrez. The 2D Dependency Pair Framework for conditional rewrite systems. Part I: Definition and basic processors. *Journal of Computer and System Sciences*, 96:74–106, 2018.

[6] W. McCune. Prover9 and Mace4. [online]. Available at https://www.cs.unm.edu/~mccune/mace4/.

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

CoLL-Saigawa 1.5: A Joint Confluence Tool*

Kiraku Shintani and Nao Hirokawa

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 at:

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

The typical usage is: `collsaigawa <file>`. Here the input file is written in the TRS format [6]. 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 v1.5 [9] and Saigawa v1.9 [2]. If an input TRS is left-linear, CoLL proves confluence. Otherwise, Saigawa analyzes confluence. CoLL is a commutation tool specialized for left-linear TRSs. It proves confluence as self-commutation by using Hindley's commutation theorem [1] together with the three commutation criteria: Almost development closeness [10], rule labeling with weight function [11], and Church-Rosser modulo A/C [4]. Saigawa can deal with non-left-linear TRSs. The tool employs the four confluence criteria: The criteria based on critical pair systems [3, Theorem 3] and on extended critical pairs [5, Theorem 2], rule labeling [11], and Church-Rosser modulo AC [4]. Furthermore, version 1.5 supports the following confluence criteria: parallel closedness based on parallel critical pairs [12], simultaneous closedness [7], and also parallel-upside and outside closedness [8].

References

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

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

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

[4] 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.

[5] 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.

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

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

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

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

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

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

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

*Supported by JSPS KAKENHI Grant Number 17K00011.

Moca 0.2: A First-Order Theorem Prover for Horn Clauses

Yusuke Oi and Nao Hirokawa

JAIST, Japan

Moca is a fully automatic first-order theorem prover for Horn clauses. The tool, written in Haskell, is freely available from:

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

The usage is: `moca.sh <file>`. Given a satisfiability problem in the TPTP CNF format [5], the tool outputs `Satisfiable` or `Unsatisfiable` if its satisfiability or unsatisfiability is proved, respectively, and `Maybe` otherwise. Given an infeasibility problem in the CoCo format [2], the tool outputs `YES` if its infeasibility is proved, and `MAYBE` otherwise.

Moca implements *maximal ordered completion* [6] and new *approximation* techniques. With a small example we illustrate how Moca uses them to solve problems. Consider the infeasibility problem of the conversion $x - x \leftrightarrow^* \mathsf{s}(x)$ for the TRS $\{x - \mathsf{0} \to x, \mathsf{0} - x \to \mathsf{0}, \mathsf{s}(x) - \mathsf{s}(y) \to x - y\}$. The problem can be regarded as the satisfiability problem of the Horn clauses:

$$x - \mathsf{0} \approx x \qquad \mathsf{0} - x \approx \mathsf{0} \qquad \mathsf{s}(x) - \mathsf{s}(y) \approx x - y \qquad x - x \not\approx \mathsf{s}(x)$$

By applying the *split-if* encoding [1] the problem reduces to the word problem of deciding $\mathsf{T} \not\approx_{\mathcal{E}} \mathsf{F}$ for the equational system $\mathcal{E}$:

$$x - \mathsf{0} \approx x \qquad \mathsf{0} - x \approx \mathsf{0} \qquad \mathsf{s}(x) - \mathsf{s}(y) \approx x - y \qquad \mathsf{f}(\mathsf{s}(x), x) \approx \mathsf{F} \qquad \mathsf{f}(x - x, x) \approx \mathsf{T}$$

In order to solve it our tool attempts to construct a ground-complete presentation of $\mathcal{E}$ by using maximal ordered completion. However, the attempt is doomed to fail as the completion diverges. Moca overcomes the divergence by approximating the last equation to the more general equation $\mathsf{f}(x - x, y) \approx \mathsf{T}$. This results in the following equational system:

$$x - \mathsf{0} \approx x \qquad \mathsf{0} - x \approx \mathsf{0} \qquad \mathsf{s}(x) - \mathsf{s}(y) \approx x - y \qquad \mathsf{f}(\mathsf{s}(x), x) \approx \mathsf{F} \qquad \mathsf{f}(x - x, y) \approx \mathsf{T}$$

Now maximal ordered completion builds up the finite ground-complete presentation $\mathcal{R}$ of the approximated equational system:

$$x - \mathsf{0} \to x \quad \mathsf{0} - x \to \mathsf{0} \quad \mathsf{s}(x) - \mathsf{s}(y) \to x - y \quad \mathsf{f}(\mathsf{0}, y) \to \mathsf{T} \quad \mathsf{f}(\mathsf{s}(x), x) \to \mathsf{F} \quad \mathsf{f}(x - x, y) \to \mathsf{T}$$

Since $\mathsf{T}{\downarrow_{\mathcal{R}}} \neq \mathsf{F}{\downarrow_{\mathcal{R}}}$ holds, infeasibility of the conversion $x - x \leftrightarrow^* \mathsf{s}(x)$ is concluded. Version 0.2 of Moca supports the generalized split-if encoding [3] and *inlining* for conditional rewrite rules [4].

References

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

[2] A. Middeldorp, J. Nagele, and K. Shintani. Confluence Competition 2019. In *Proc. 25th TACAS (Part III)*, *LNCS* 11429, pp. 25–40, 2019.

[3] Y. Oi. Refutation by Completion and Approximations. Master's thesis, JAIST, 2019.

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

[5] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

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

CoCo 2020 Participant: CSI 1.2.4

Fabian Mitterwallner and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria
fabian.mitterwallner@uibk.ac.at, aart.middeldorp@uibk.ac.at

CSI is an automatic tool for (dis)proving confluence and related properties of first-order term rewrite systems (TRSs). It has been in development since 2010. Its name is derived from the Confluence of the rivers Sill and Inn in Innsbruck. The tool is available from

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

under a LGPLv3 license. A detailed description of CSI can be found in [3]. Some of the implemented techniques are described in [1, 2, 4]. Compared to last year's version, CSI 1.2.4 contains an implementation of the upside-parallel-closure criterion for confluence by Oyamaguchi and Ohta [5]. Additionally some minor changes to the strategy have been made.

CSI participates in the following CoCo 2020 categories: CPF-TRS, NFP, SRS, TRS, UNC, and UNR.

References

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

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

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

[5] M. Oyamaguchi and Y. Ohta. A New Parallel Closed Condition for Church-Rosser of Left-Linear Term Rewriting Systems. In *Proc. 8th International Conference on Rewriting Techniques and Applications*, volume 1232 of *Lecture Notes in Computer Science*, pages 187–201, 1997. doi: 10.1007/3-540-62950-5_70

CoCo 2020 Participant: FORT-h 0.9[*]

Fabian Mitterwallner, Aart Middeldorp, and Bertram Felgenhauer

Department of Computer Science, University of Innsbruck, Austria
fabian.mitterwallner@uibk.ac.at, aart.middeldorp@uibk.ac.at, int-e@gmx.de

The first-order theory of rewriting is a decidable theory for finite left-linear right-ground rewrite systems. The decision procedure goes back to Dauchet and Tison [1] and is implemented in FORT [3,4]. In this theory confluence-related properties on ground terms are easily expressible.

FORT-h implements a new variant, described in [2], of the decision procedure for the larger class of linear variable-separated rewrite systems. This variant supports a more expressive theory and is based on anchored ground tree transducers. FORT-h 0.9 is implemented in Haskell. A command-line version of the tool can be downloaded from

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

FORT-h participates in the following CoCo 2020 categories: COM, GCR, NFP, UNC, and UNR. A future release of FORT-h will produce certificates for the YES/NO answers that will be checked by an independent certifier based on the Isabelle/HOL formalization described in [2].

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] B. Felgenhauer, A. Lochmann, A. Middeldorp, and F. Mitterwallner. Formalizing the First-Order Theory of Rewriting. Submitted for publication, 2020.

[3] 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.

[4] F. Rapp and A. Middeldorp. FORT 2.0. In *Proc. 9th International Joint Conference on Automated Reasoning*, volume 10900 of *LNCS (LNAI)*, pages 81–88, 2018. doi: 10.1007/978-3-319-94205-6_6.

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

CoCo 2020 Participant: ConCon 1.10

Christian Sternagel

DVT, Innsbruck, Austria

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 at

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

For more details on its implementation and employed methods we refer to an earlier system description [5].

Starting from version 1.10 ConCon supports arbitrary external tools for proving infeasibility. In CoCo 2020 this will be showcased by using nonreach (version 1.2.2) [3] as external tool.

Other external tools that are used to discharge subgoals concerning (non-conditional) confluence and termination are CSI (version 1.2.3) [2] and $\mathsf{T_TT_2}$ (version 1.20) [1].

While most other methods of ConCon can be certified using CeTA (version 2.39) [4, 6], certification of nonreach proofs is future work.

References

[1] Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2 In *Proc. 20th RTA*, 2009, doi:10.1007/978-3-642-02348-4_21

[2] Bertram Felgenhauer, Aart Middeldorp, and Fabian Mitterwallner. CoCo 2019 Participant: CSI 1.2.3. In *Proc. 8th IWC*, 2019. `http://iwc2019.cic.unb.br/proc-HOR-IWC-CoCo.pdf`

[3] F. Meßner. CoCo 2020 Participant: nonreach. In *Proc. 9th IWC*, 2020. To appear.

[4] J. Schöpf, C. Sternagel, R. Thiemann, and A. Yamada. CoCo 2020 Participant: CeTA 2.39. In *Proc. 9th IWC*, 2020. To appear.

[5] C. Sternagel and S. Winkler. CoCo 2019 Participant: ConCon 1.9. In *Proc. 8th IWC*, 2019. `http://iwc2019.cic.unb.br/proc-HOR-IWC-CoCo.pdf`

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

CO3 (Version 2.1)

Naoki Nishida
Nagoya University, Nagoya, Japan
`nishida@i.nagoya-u.ac.jp`

CO3, a converter for proving confluence of conditional TRSs,[1] tries to prove confluence of conditional term rewriting systems (CTRSs, for short) by using a transformational approach (cf. [4]). The tool first transforms a given weakly-left-linear (WLL, for short) 3-DCTRS into an unconditional term rewriting system (TRS, for short) by using $\mathbb{U}_{\mathit{conf}}$ [2], a variant of the *unraveling* $\mathbb{U}$ [6], and then verifies confluence of the transformed TRS by using the following theorem: a 3-DCTRS $\mathcal{R}$ is confluent if $\mathcal{R}$ is WLL and $\mathbb{U}_{\mathit{conf}}(\mathcal{R})$ is confluent [1, 2]. The tool is very efficient because of very simple and lightweight functions to verify properties such as confluence and termination of TRSs. Since version 2.0, a *narrowing-tree*-based approach [5, 3] to prove infeasibility of a condition w.r.t. a specified CTRS has been implemented. The approach is applicable to *syntactically deterministic* CTRSs that are operationally terminating and *ultra-right-linear* w.r.t. the *optimized* unraveling. In the present version, bugs in version 2.0 has been fixed and the computation of SCCs for termination has slightly been improved.

To prove confluence by means of narrowing trees, the tool first computes the (conditional) critical pairs, and then proves their joinability as follows: a critical pair $\langle s, t\rangle \Leftarrow c$ is joinable if (1) c is the empty list and $s = t$, or (2) the narrowing tree for c can be simplified to a tree that defines the empty set of substitutions. For example, let us consider `489.trs` in Cops which is an operationally terminating normal 1-CTRS, and has a conditional critical pair $\langle \mathsf{true}, \mathsf{false}\rangle \Leftarrow \mathsf{o}(x) \twoheadrightarrow \mathsf{true}, \mathsf{e}(x) \twoheadrightarrow \mathsf{true}$. As a narrowing tree for condition $\mathsf{o}(x) \twoheadrightarrow \mathsf{true}, \mathsf{e}(x) \twoheadrightarrow \mathsf{true}$ w.r.t. `489.trs`, we construct the following production rules for a regular tree grammar [5]:

$$\begin{aligned}
\Gamma_{\mathsf{e}(x)\twoheadrightarrow\mathsf{true}\,\&\,\mathsf{o}(x)\twoheadrightarrow\mathsf{true}} &\to \textsc{Rec}(\Gamma_{\mathsf{e}(x')\twoheadrightarrow\mathsf{true}}, \{x \mapsto x'\}) \,\&\, \textsc{Rec}(\Gamma_{\mathsf{o}(x'')\twoheadrightarrow\mathsf{true}}, \{x \mapsto x''\}) \\
\Gamma_{\mathsf{e}(x')\twoheadrightarrow\mathsf{true}} &\to \mathit{id} \,\&\{x' \mapsto \mathsf{0}\} \mid \big(\textsc{Rec}(\Gamma_{\mathsf{o}(x'')\twoheadrightarrow\mathsf{true}}, \{x_1 \mapsto x''\}) \,\&\, \mathit{id}\big) \,\&\{x' \mapsto \mathsf{s}(x_1)\} \\
&\quad \mid \big(\textsc{Rec}(\Gamma_{\mathsf{e}(x')\twoheadrightarrow\mathsf{true}}, \{x_2 \mapsto x'\}) \,\&\, \varnothing\big) \,\&\{x' \mapsto \mathsf{s}(x_2)\} \\
\Gamma_{\mathsf{o}(x'')\twoheadrightarrow\mathsf{true}} &\to \varnothing \,\&\{x'' \mapsto \mathsf{0}\} \mid \big(\textsc{Rec}(\Gamma_{\mathsf{e}(x')\twoheadrightarrow\mathsf{true}}, \{x_3 \mapsto x'\}) \,\&\, \mathit{id}\big) \,\&\{x'' \mapsto \mathsf{s}(x_3)\} \\
&\quad \mid \big(\textsc{Rec}(\Gamma_{\mathsf{o}(x'')\twoheadrightarrow\mathsf{true}}, \{x_4 \mapsto x''\}) \,\&\, \varnothing\big) \,\&\{x'' \mapsto \mathsf{s}(x_4)\}
\end{aligned}$$

These rules can be simplified to $\Gamma_{\mathsf{e}(x)\twoheadrightarrow\mathsf{true}\,\&\,\mathsf{o}(x)\twoheadrightarrow\mathsf{true}} \to \varnothing$, and the critical pair is infeasible.

To prove infeasibility of a condition c, the tool first prove confluence, and then linearizes c if failed to prove confluence; then, the tool computes and simplifies a narrowing tree for c, and examines the emptiness of the narrowing tree.

References

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

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

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

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

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

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

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

CoLL 1.5: A Commutation Tool

Kiraku Shintani

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

CoLL (version 1.5) is a tool for automatically proving commutation of left-linear term rewrite systems (TRSs). The tool, written in OCaml, is freely available at:

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

The typical usage is: `coll <file>`. Here the input file is written in the commutation problem format [10]. The tool outputs `YES` if commutation of the input TRSs is proved, `NO` if non-commutation is shown, and `MAYBE` if the tool does not reach any conclusion.

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

Theorem 1 ([3]). *ARSs $\mathcal{A} = \langle A, \{\to_\alpha\}_{\alpha \in I} \rangle$ and $\mathcal{B} = \langle A, \{\to_\beta\}_{\beta \in J} \rangle$ commute if $\to_\alpha$ and $\to_\beta$ commute for all $\alpha \in I$ and $\beta \in J$.*

Here indexes are interpreted as subsystems of the input TRSs. For every pair of subsystems the tool proves the commutation property, employing the three criteria: Development closeness [2, 7], rule labeling with weight function [8, 1], and Church-Rosser modulo A/C [4]. A detailed description of CoLL can be found in [6].

As a final remark, the bug of AC-related method, reported in [9], has been fixed in the current version of CoLL.

References

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

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

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

[4] 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.

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

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

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

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

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

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

CoCo 2020 Participant: nonreach[*]

Florian Meßner

University of Innsbruck, Innsbruck, Austria florian.g.messner@uibk.ac.at

The tool nonreach is an automated, efficient tool to check infeasibility with respect to oriented conditional term rewrite systems (CTRSs). The Haskell source code can be obtained from a public *git* repository hosted on *bitbucket*:

https://bitbucket.org/fmessner/nonreach

Given a CTRS (or a TRS) and one or more infeasibility problems, nonreach uses a combination of *decomposition*, based on narrowing (with some heuristics) and proving root-nonreachability [2], and *fast checks*, based on etcap [3] and the *inductive symbol transition graph* [2].

These methods are applied by turns until I either obtain infeasibility (by simplifying the tree to False), a satisfying substitution or reach a user-defined threshold of iterations (and nonreach concludes MAYBE).

I outline the main new features of nonreach 1.2 compared to the version participating in last year's CoCo.

- *Certification* of (some) proofs (which is not visible in the competition for the lack of a CPF-INF category).
- *Positive reachability results* found through narrowing now yield NO together with a satisfying assignment.

While refactoring was necessary in order to generate certificates, and as a nice side-effect leads to more detailed and more readable proofs, I lose a few infeasibility results compared to last year. Furthermore, after finding a bug in internal meetability problem handling, which in rare cases could lead to unsound results, I disabled almost all of those methods, thus losing a few more infeasibility results.

References

[1] Florian Meßner and Christian Sternagel. nonreach - A tool for nonreachability analysis. In *Proc. 25th TACAS*, pages 337–343, 2019. doi:10.1007/978-3-030-17462-0_19.

[2] Christian Sternagel and Akihisa Yamada. Reachability analysis for termination and confluence of rewriting. In *Proc. 25th TACAS*, pages 262–278, 2019. doi:10.1007/978-3-030-17462-0_15.

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

[*]This work is supported by the Austrian Science Fund (FWF): project P27502.

ACP: System Description for CoCo 2020

Takahito Aoto[1]

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

A primary functionality of ACP is proving confluence (CR) of term rewriting systems (TRSs). ACP integrates multiple direct criteria for guaranteeing confluence of TRSs. It also incorporates divide–and–conquer criteria by which confluence or non-confluence of TRSs can be inferred from those of their components. Several methods for disproving confluence are also employed. For some criteria, it supports generation of proofs in CPF format that can be certified by certifiers. The internal structure of the prover is kept simple and is mostly inherited from the version 0.11a, which has been described in [3]. It also deal with confluence of oriented conditional term rewriting systems. Besides confluence, ACP now supports proving the UNC property (unique normal form property w.r.t. conversion) and the commutation property of term rewriting systems. The ingredients of the former property have been appeared in [2, 4]. Our (dis)proofs of commutation are based on a development closed criterion [5] and a simple search for counter examples. No new (CR/UNC) criterion has been incorporated from the one submitted for CoCo 2019.

ACP is written in Standard ML of New Jersey (SML/NJ) and the source code is also available from [1]. It uses a SAT prover such as MiniSAT and an SMT prover YICES as external provers. It internally contains an automated (relative) termination prover for TRSs but external (relative) termination provers can be substituted optionally. Users can specify criteria to be used so that each criterion or any combination of them can be tested. Several levels of verbosity are available for the output so that users can investigate details of the employed approximations for each criterion or can get only the final result of prover's attempt.

References

[1] ACP (Automated Confluence Prover). http://www.nue.ie.niigata-u.ac.jp/tools/acp/.

[2] T. Aoto and Y. Toyama. Automated proofs of unique normal forms w.r.t. conversion for term rewriting systems. In *Proc. of 12th FroCoS*, volume 11715 of *LNAI*, pages 330–347. Springer-Verlag, 2019.

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

[4] M. Yamaguchi and T. Aoto, A fast decision procedure for uniqueness of normal forms w.r.t. conversion of shallow term rewriting systems. In *Proc. of 5th FSCD*, volume 167 of *LIPIcs*, pages 9:1–9:23. Schloss Dagstuhl, 2020.

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

AGCP: System Description for CoCo 2020

Takahito Aoto

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

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

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

No new ground (non-)confluence criterion has been incorporated from the one submitted for CoCo 2019.

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 2020 Participant: CSI^ho 0.3.2

Julian Nagele

London, UK
mail@jnagele.net

CSI^ho is a tool for automatically (dis)proving confluence of higher-order rewrite systems, specifically pattern rewrite systems (PRSs) as introduced by Nipkow [1,4]. CSI^ho is an extension of CSI, a confluence prover for first-order rewrite systems.

No new features were added to CSI^ho since CoCo 2018—it ran unopposed in the HRS category of CoCo 2019. A detailed description of CSI^ho be found in [2,3] or earlier CoCo system descriptions. The tool is available at

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

References

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

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

[3] Julian Nagele, Bertram Felgenhauer, and Aart Middeldorp. CSI: New evidence — A progress report. In *Proc. 26th CADE*, volume 10395 of *LNCS (LNAI)*, pages 385–397, 2017.

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

CoCo 2020 Participant: CeTA 2.39*

Jonas Schöpf[1], Christian Sternagel[2], René Thiemann[1], and Akihisa Yamada[3]

[1] University of Innsbruck, Austria
[2] DVT, Austria
[3] National Institute of Advanced Industrial Science and Technology, Japan

The tool CeTA [2] is a certifier for, among other properties, (non-)confluence of term rewrite systems with and without conditions. Its soundness is proven as part of the formal proof library IsaFoR, the Isabelle Formalization of Rewriting. For a complete reference of supported techniques we refer to the certification problem format (CPF) and the IsaFoR/CeTA website:

http://cl-informatik.uibk.ac.at/isafor/

In the following, we describe what is new in version 2.39 of CeTA. Although there are no new techniques in CeTA that are specific for confluence proving, we like to mention two newly supported termination methods [3]. Both of these extensions have the potential to increase the power of confluence techniques that rely upon termination or relative termination.

The first extension consists of support for the weighted path order (WPO) [5], a term order that unifies and extends well-known path orders such as the Knuth–Bendix order and the lexicographic path order. In particular, confluence provers can now for the first time use NaTT [4] – which is specialized on WPO – as external termination prover in order to produce certifiable confluence proofs.

The second extension is the support for max-polynomial interpretations [1], i.e., polynomial interpretations that additionally allow the maximum operator. In CeTA these orders can be used stand-alone, but also in combination with WPO.

We would like to welcome all confluence tool developers to experiment with whether our two extensions are indeed helpful for confluence proving, and are looking forward to certify these new kinds of proofs via CeTA.

References

[1] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. Maximal Termination. In *Rewriting Techniques and Applications, 19th International Conference, Proceedings*, volume 5117 of *LNCS*, pages 110–125. Springer, 2008.

[2] René Thiemann and Christian Sternagel. Certification of Termination Proofs Using CeTA. In *Theorem Proving in Higher Order Logics, 22nd International Conference, Proceedings*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009.

[3] René Thiemann, Jonas Schöpf, Christian Sternagel, and Akihisa Yamada. Certifying the Weighted Path Order. In *Formal Structures for Computation and Deduction, 5th International Conference, Proceedings*, 2020. To appear.

[4] Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. Nagoya Termination Tool. In *Rewriting and Typed Lambda Calculi - Joint International Conference, Proceedings*, volume 8560 of *LNCS*, pages 466–475. Springer, 2014.

[5] Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. A Unified Ordering for Termination Proving. *Sci. Comput. Program.*, 111:110–134, 2015.

*Jonas Schöpf and Christian Sternagel were supported by FWF project P27502. René Thiemann was supported by the Austrian Science Fund (FWF) project Y757. A part of Akihisa Yamada's contribution was made while working at the University of Innsbruck supported by FWF projects Y757 and P27502.

The System SOL version 2020

Makoto Hamana[1], Kentaro Kikuchi[2],
Date Yao Faustin Dieudonne[1], Kazuki Fuju[1]

[1] Department of Computer Science, Gunma University, Japan
hamana@cs.gunma-u.ac.jp
[2] RIEC, Tohoku University, Japan
kentaro.kikuchi@riec.tohoku.ac.jp

SOL is a Haskell-based tool for confluence and strong normalisation of higher-order computation. SOL is intended to be a generic higher-order computation analysis tool that is applicable to the modern theories of higher-order programming languages. This aim is demonstrated in [Ham19] and further developed in [Ham18].

Based on the foundation of second-order algebraic theories [FH10] and its computational counter part [Ham19] and polymorphic extension [Ham18], we implemented various results on higher-order syntax and computation in SOL, including Knuth and Bendix's critical pair checking for confluence, and Function-as-Constructor Unification (FCU) [LM16] for unification. Termination analysis is based on the General Schema criterion [Bla00, Bla16].

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.

[Bla16] F. Blanqui. Termination of rewrite relations on λ-terms based on Girard's notion of reducibility. *Theor. Comput. Sci.*, 611:50–86, 2016.

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

[Ham19] M. Hamana. How to prove decidability of equational theories with second-order computation analyser SOL. Journal of Functional Programming, Cambride University Press, Vol. 29, e20, 2019.

[Ham18] M. Hamana. Polymorphic Rewrite Rules: Confluence, Type Inference, and Instance Validation, *Functional and Logic Programming (FLOPS'18)*, Lecture Notes in Computer Science 10818, pp.99-115, Springer, 2018.

[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.

Author Index