Proceedings of the

10th International Workshop on Confluence

July 23rd 2021

Online

Foreword

This report contains the proceedings of the 10th International Workshop on Confluence (IWC) which took place on July 23rd, 2021. It was due to be held in Buenos Aires, Argentina, 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 10th Confluence Competition (CoCo 2021). The workshop was co-located with the FSCD 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 7 contributed talks as well as invited talks by Jesper Cockx and José Meseguer. In addition, the program contains the system descriptions from the 10th Confluence Competition (CoCo 2021). 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 for hosting the workshops.

June 15th, 2021, Paris
Samuel Mimram
Camilo Rocha

Steering Committee

- Takahito Aoto
- Mauricio Ayala-Rincón

Program Committee

- Beniamino Accattoli (INRIA & LIX, École Polytechnique)
- Sandra Alves (Universidade do Porto)
- Cyrille Chenavier (Université de Limoges)
- Francisco Durán (University of Málaga)
- Alejandro Díaz-Caro (Universidad Nacional de Quilmes & ICC/UBA-CONICET)
- Samuel Mimram (LIX, École Polytechnique), co-chair
- Camilo Rocha (Pontificia Universidad Javeriana), co-chair
- Femke van Raamsdonk (VU University Amsterdam)
- Sarah Winkler (University of Bolzano)

Additional reviewers

- Eduardo Bonelli
- Rafael Romero

Contents

Multi-redexes and multi-treks induce residual systems

least upper bounds and left-cancellation up to homotopy

Vincent van Oostrom

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

Abstract

Residual theory in rewriting goes back to Church, Rosser and Newman at the end of the 1930s. We investigate an axiomatic approach to it developed in 2002 by Melliès. He gave four axioms (SD) *self-destruction*, (F) *finiteness*, (FD) *finite developments*, and (PERM) *permutation*, showing that they entail two key properties of reductions, namely having (i) *least upper bounds* (*lubs*) and (ii) *left-cancellation*.[1] These properties are shown to hold *up to* the equivalence generated by identifying the legs of local confluence diagrams inducing the same residuation, which corresponds to Lévy's permutation equivalence. Melliès in fact presented two sets of axioms, one for *redexes* as in classical residual theory and another more general one for *treks*. We show his results factor through the theory of *residual* systems we introduced in 2000, in that any rewrite system satisfying the four axioms (for redexes or treks) can be enriched to a residual system such that (i) and (ii) follow from the theory of residual systems. We exemplify the axioms are sufficient but not necessary.

Proofs omitted in this abstract can be found in the appendix of [18].

1 Residual systems

We are interested in the theory of computation based on rewriting. As this requires to have computations as first-class citizens, we use rewrite *systems* [14],[20, Def. 8.2.2] (not rewrite *relations*), whose *steps* have *sources* and *targets*. We recapitulate *residual* systems [20, Def. 8.7.2].

Definition 1. *A* residual *system (RS)* $\langle \to, 1, / \rangle$ *comprises a rewrite system* $\to$ *and a* residual *function* $/$ *having* 1 *as* unit*:* 1 *is a function from objects to steps such that* $\mathsf{tgt}(1_a) = a = \mathsf{src}(1_a)$ *and for co-initial steps* ϕ, ψ, χ*, the* residual *identities (1)–(3) in Tab. 1 must be satisfied. The* projection *order* $\lesssim$ *is defined by* $\phi \lesssim \psi$ *if* $\phi/\psi = 1$ *for co-initial steps* ϕ, ψ*.*

The projection order $\lesssim$ is a quasi-order [20, Lem. 8.7.23] inducing *projection* equivalence $\simeq := \lesssim \cap \gtrsim$. Examples of rewrite systems that can be equipped with residual structure abound.

Example 1. *For the following rewrite systems* $\to$*, residual structure is obtained from the proof of the diamond property for an appropriate rewrite system that is between* $\to$ *and its reflexive–transitive closure: i) the* $\lambda\beta$*-calculus induces a residual system by the Tait–Martin-Löf proof that* $\geq_1$ *has the diamond property [1]; ii)* β*-steps in the* linear $\lambda\beta$*-calculus have the diamond property themselves; iii)* parallel *steps* $\twoheadrightarrow$/multisteps $\multimap$ *in orthogonal first/higher-order term rewrite systems [8, 20, 2]; iv) positive braids with* parallel crossings *of strands [20, Sect. 8.9].*

Here we show multi-redexes and multi-treks as in Melliès' axiomatic residual theory naturally induce residual systems, entailing the results of [13] *via* the theory of residual systems [20]. We use $\phi, \psi, \chi, \ldots$ and $\gamma, \delta, \epsilon, \ldots$ to range over steps respectively reductions. We denote *finite*

[1] Instead of the order-theoretic setting employed here, Melliès employs a category-theoretic setting and the corresponding terminology of having *pushouts* and *epis*.

reductions by $\twoheadrightarrow$. They can be identified [20, Def. 8.2.10] with formal compositions $(\cdot)$ of steps (whose targets, sources match) modulo the monoid *identities*. Orienting these into the *rules* (4)–(6) of Tab. 1 gives a complete 2-rewrite system[2] so unique representatives of such reductions.

Proposition 1. *Any residual system on $\to$ extends to a residual system on $\twoheadrightarrow$, defining residuation by normalisation w.r.t. the* 2*-rewrite system with rules (4)–(8) of Tab. 1.*

$$\begin{array}{rcl} \phi/1 & \overset{(1)}{=} & \phi \\ \phi/\phi & \overset{(2)}{=} & 1 \\ (\phi/\psi)/(\chi/\psi) & \overset{(3)}{=} & (\phi/\chi)/(\psi/\chi) \end{array} \qquad \begin{array}{rcl} (\gamma\cdot\delta)\cdot\epsilon & \overset{(4)}{\Rightarrow} & \gamma\cdot(\delta\cdot\epsilon) \\ \gamma\cdot 1 & \overset{(5)}{\Rightarrow} & \gamma \\ 1\cdot\gamma & \overset{(6)}{\Rightarrow} & \gamma \end{array} \qquad \begin{array}{rcl} \gamma/(\delta\cdot\epsilon) & \overset{(7)}{\Rightarrow} & (\gamma/\delta)/\epsilon \\ (\delta\cdot\epsilon)/\gamma & \overset{(8)}{\Rightarrow} & (\delta/\gamma)\cdot(\epsilon/(\gamma/\delta)) \end{array}$$

Table 1: Residual identities, monoid rules, and residual rules for formal composition

Example 2. *The classical example of a term rewrite system is Combinatory Logic (CL) having the three rules, in applicative notation, $\iota(x):Ix\to x$, $\kappa(x,y):Kxy\to x$, and $\varsigma(x,y,z):Sxyz\to xz(yz)$. We call a term over the signature extended with the so-called* rule *symbols [20, Ch. 8] ι,κ,ς (having as arities the number of variables in the respective rules) a* multistep*, as it can be assigned a source/target by mapping all such rule symbols in it to their lhs/rhs. This naturally induces a residual system on multisteps [20, Prop. 8.7.7], which by the above extends to one on reductions (of multisteps). For example, $\gamma:=\varsigma(K,y,Iz)\cdot\kappa(Iz,y(Iz))$ and $\delta:=SKI\iota(z)$ are co-initial reductions from $SKy(Iz)$ to Iz respectively $SKyz$. Both these targets are reduced to z by the respective residual reductions: $\delta/\gamma:=\iota(z)$ and $\gamma/\delta:=\varsigma(K,y,z)\cdot\kappa(z,y(z))$.*

Remark 1. *We introduced the idea of multisteps as terms over the signature extended with rule symbols in [20, Ch. 8] as a generic tool in* structured *rewrite systems, like* string *[6, p. 226],* higher-order term *[2, p. 127], and* graph *[20, Rem. 9.4.30] rewrite systems.*

Then $\twoheadrightarrow$ is a residual system *with composition* [20, Def. 8.7.38], $\simeq$ is a congruence for $/$ and $\cdot$ and quotienting $\simeq$ out yields a residual system whose projection order is a partial order [20, Lem. 8.7.41]. Projection equivalence [20] can alternatively be defined as the homotopy *generated* by the diamond property. This will allow us below to relate the former to local homotopy [13].

Definition 2. Square homotopy *equivalence $\equiv$ on reductions having the same sources/targets, is generated by closing $\phi\sqcup\psi\equiv\psi\sqcup\phi$ for local peaks ϕ,ψ under composition: if $\gamma\equiv\gamma'$ then $\delta\cdot\gamma\cdot\epsilon\equiv\delta\cdot\gamma'\cdot\epsilon$. Here $\phi\sqcup\psi:=\phi\cdot(\psi/\phi)$. Correspondingly, we define $\gamma\sqsubseteq\delta$ if $\gamma\cdot\epsilon\equiv\delta$ for some ϵ.*

Lemma 1. *$\simeq\,=\,\equiv$ and $\lesssim\,=\,\sqsubseteq$.*

Example 3. *For γ,δ in Ex. 2 we have $\gamma\cdot(\delta/\gamma)\equiv\varsigma(K,y,Iz)\cdot K(\iota(z))(y\iota(z))\cdot\kappa(z,yz)\equiv\delta\cdot(\gamma/\delta)$.*

Theorem 1. *$\twoheadrightarrow$ up to square homotopy has lubs (δ',γ' is an* upper bound *of γ,δ if $\gamma\cdot\delta'\equiv\delta\cdot\gamma'$;* least *if $\delta'\sqsubseteq\delta'',\gamma'\sqsubseteq\gamma''$ for all upper bounds δ'',γ'') and left-cancellation (if $\gamma\cdot\delta\equiv\gamma\cdot\epsilon$ then $\delta\equiv\epsilon$).*

[2] What we refer to as 2-rewrite systems have formal expressions of compositions (and residuations) as objects. Their rules transform such expressions into *reductions* of an ordinary (1-)rewrite system $\to$, i.e. into formal compositions in normal form with respect to the monoid rules. This set-up generalises the 2-rewrite systems as found in the literature by not giving special status to composition, not *assuming* rules to operate on reductions only but on formal expressions. Working *modulo* the monoid identities yields proper 2-rewrite systems.

Proof. By Lem. 1 it follows from the same for projection equivalence $\simeq$ instead of square homotopy $\equiv$, which holds by virtue of $\twoheadrightarrow$ being a residual system with composition [20, Ex. 8.7.52]. We do that exercise: Left-cancellation follows from (see also the proof of Prop. 1):

$$(\gamma\cdot\delta)/(\gamma\cdot\epsilon) \Rightarrow ((\gamma\cdot\delta)/\gamma)/\epsilon \Rightarrow ((\gamma/\gamma)\cdot(\delta/(\gamma/\gamma)))/\epsilon \twoheadrightarrow (1\cdot(\delta/1))/\epsilon \twoheadrightarrow (1\cdot\delta)/\epsilon \Rightarrow \delta/\epsilon$$

That $\delta/\gamma, \gamma/\delta$ is an upper bound up to $\simeq$ of γ,δ, holds by $\twoheadrightarrow$ being a residual system. To see it is least consider any δ'',γ'' such that $\gamma\cdot\delta''\simeq\delta\cdot\gamma''$. Then $(\gamma\cdot\delta'')/(\delta\cdot\gamma'') \Rightarrow (\gamma/(\delta\cdot\gamma''))\cdot(\delta''/((\delta\cdot\gamma'')/\gamma)) = 1$. Therefore [20, Ex. 8.7.40(iii)] both components must be 1 in particular the 1st $\gamma/(\delta\cdot\gamma'') \Rightarrow (\gamma/\delta)/\gamma'' = 1$. By symmetry $(\delta/\gamma)/\delta'' = 1$ and we conclude.[3] □

2 Multi-redexes and multi-treks

In [13] rewrite systems are equipped with a notion of residuation inducing a notion of *local* homotopy on reductions, based on the four axiomatic properties (SD), (F), (FD), and (PERM). The properties guarantee that multi-redexes/treks can be *developed* into reductions, that such developments have the diamond property, that all developments are locally homotopic, and finally (the main result) that reductions have lubs and left-cancellation up to local homotopy (Thm. 2). In fact *two* sets of four axioms are given in [13], the first one for *multi-redexes* and the second more general one for *multi-treks*. We show that in both cases the main results of [13] follow by known residual theory for a naturally associated residual system (in the sense of Sect. 1) on so-called developments, in particular from Thm. 1. We first develop enough notation to formally *express* the properties required of a rewrite system $\to$ for *multi-redexes* [13, Section 2], which informally read:

(*self-destruction*, SD) no step has a residual after itself;

(*finiteness*, F) every redex has finitely many residuals after a step;

(*finite developments*, FD) developments of multi-redexes are finite; and

(*permutation*, PERM) every peak ϕ,ψ of steps can be completed by a valley of complete developments of the residuals of ψ after ϕ, respectively the residuals of ϕ after ψ, such that both legs of the resulting local confluence diagram induce the same redex-trace relation.

We then show that these properties induce a residual system (Def. 1) on developments whose square homotopy corresponds to local homotopy on reductions, i.e. that Thm. 1 entails Thm. 2:

Theorem 2 (SD,FD,PERM; [13])**.** *$\twoheadrightarrow$ up to local homotopy has lubs and left-cancellation.*

Here *local* homotopy is generated (Def. 5) from the *local* confluence diagrams given by (PERM), instead of the *square* diamonds generating *square* homotopy (Def. 2). As in the statement of this main theorem, we qualify (intermediate) results throughout with the properties used, to enable illustrating that properties are sufficient but not necessary. In [13] residuation is captured by *tracing* a *redex* along a step to its *residuals*.

Definition 3. *A* redex-*trace relation is a function $[\![\cdot]\!\rangle$ mapping each step $\phi\colon a\to b$ to a relation $[\![\phi]\!\rangle$ between the redexes of a and b, where (*multi-*)*redexes *are reified (sets of) steps.*

[3] That gives a *pushout* as witnessed by $\epsilon := \delta''/(\delta/\gamma)$: On the one hand, $(\delta/\gamma)\cdot\epsilon\simeq\delta'\cdot((\delta/\gamma)/\delta'')\simeq\delta''$ follows from having a residual system and $\delta/\gamma\lesssim\delta'$. On the other hand, $(\gamma/\delta)\cdot\epsilon\simeq\epsilon''$ follows by left-cancellation from $\delta\cdot(\gamma/\delta)\cdot\epsilon\simeq\gamma\cdot(\delta/\gamma)\cdot\epsilon\simeq\gamma\cdot\delta''\simeq\delta\cdot\epsilon''$ where the 2nd equivalence holds by the above and the others by assumption.

(SD) is formalised as $(\phi\ [\![\phi]\!]) = \varnothing$ and (F) as $(\psi\ [\![\phi]\!])$ is finite, for any step ϕ and redex ψ. Here we use *section* notation for partial application of relations. The *left* section of a binary relation for an object a is $(a\ R) := \{b \mid a\ R\ b\}$. Similarly, the *right* section is $(R\ a) := \{b \mid b\ R\ a\}$. This is lifted pointwise to sets by $(A\ R) := \bigcup_{a\in A}(a\ R)$ and $(R\ A) := \bigcup_{a\in A}(R\ a)$. Trace relations naturally extend to reductions and conversions since relations constitute an involutive (typed) monoid with respect to composition, the identity relation, and converse, so we may e.g. write $\langle\!\langle\leftarrow]\!]$ for the trace relation of $\leftarrow$. We proceed with reifying tracing, labelling objects of the rewrite system with sets of redexes, which allows to recover the notion of *development* of [13].

Definition 4. *Consider the labelled rewrite system [20, Def. 8.4.5] having for each set Φ of redexes of a the object a^Φ, and for each step $\phi{:}a \to b$ the step ϕ^Φ from a^Φ to $b^{(\Phi[\![\phi]\!])}$. A reduction γ from an object a is a* development of Φ *if it lifts to a $[\![\to]\!\rangle$-reduction γ^Φ from a^Φ, where $[\![\to]\!\rangle$ is the restriction of the labelled rewrite system to steps ϕ^Φ such that $\phi \in \Phi$. We say γ is a* complete *development of Φ if its lifting ends in a $\varnothing$-labelled object.*

(FD) is formalised by all developments are finite,[4] and (PERM) by every local peak ϕ, ψ is completed by some valley γ, δ of complete developments of $(\psi\ [\![\phi]\!]), (\phi\ [\![\psi]\!])$ with $[\![\phi{\cdot}\gamma]\!] = [\![\psi{\cdot}\delta]\!]$.

Remark 2. *The lifting γ^Φ of the reduction γ in Def. 4 is unique. Formally, this is a consequence of the labelling given being a* rewrite *labelling in the sense of [20, Def. 8.4.5].*

Lemma 2 (FD,PERM)**.** *$\langle \twoheadrightarrow, 1, / \rangle$ is a residual system with binary joins/diagonals, for $\twoheadrightarrow$ the rewrite system having as objects the objects of $\to$, and as steps a* multi-redex *$a^\Phi : a \twoheadrightarrow b$ if there is a complete development of Φ from a to b; 1_a defined as $\varnothing$; residual Φ/Ψ defined as $(\Phi\ [\![\Psi]\!])$, and the binary join/diagonal given by $\Phi \cup \Psi$ (cf. [20, Def. 8.7.28]).*

Denoting a multi-redex a^Φ by just Φ in the lemma, is justified by that a is the source common to all steps in Φ, and that all complete developments of Φ have the same target. The join being a step from the source to the target of a residual diamond, justifies calling it a diagonal.

Remark 3. *Parallel rewriting $\twoheadrightarrow\!\!\!\!\!\!+$ [7] does constitute a residual system for orthogonal TRSs, so does give rise to good residual theory [20], but $\twoheadrightarrow\!\!\!\!\!\!+$ does not have joins, e.g. the join of the single/parallel steps $\iota(Ix)$ and $I\iota(x)$* should *be $\iota(\iota(x))$ but although that is a* multi*step it is not a* parallel *step as it nests ι in itself. Hence, by Lem. 2 it cannot be obtained* via *multi-redexes; a first indication that the properties in [13] are too strong.*[5]

Definition 5 ([13])**.** Local homotopy $\equiv_l$ *on reductions with the same sources/targets, is the equivalence generated by closing $\phi \cdot \gamma \equiv_l \psi \cdot \delta$ for peaks ϕ, ψ and valleys γ, δ given*[6] *by (PERM) under composition: if $\gamma \equiv_l \gamma'$ then $\delta' \cdot \gamma \cdot \epsilon' \equiv_l \delta' \cdot \gamma' \cdot \epsilon'$. We define $\gamma \sqsubseteq_l \delta$ if $\gamma \cdot \epsilon \equiv_l \delta$ for some ϵ.*

We show *local* homotopy $\equiv_l$ on finite $\to$-reductions is the same as *square* homotopy $\equiv$ on finite $\twoheadrightarrow$-reductions. Observe we may embed $\to \subseteq \twoheadrightarrow$ by mapping a step $\phi : \phi \to \psi$ to $\phi^{\{\phi\}} : a \twoheadrightarrow b$ assuming (SD), and vice versa $\twoheadrightarrow \subseteq \twoheadrightarrow\!\!\!\!\!\!+$ by mapping each multi-redex a^Φ to an arbitrary but fixed complete development of Φ from a. Below the corresponding coercions (and their stepwise

[4] Since in [13] only *finite* reductions are defined, (FD) is (must be) circumscribed there as the absence of infinite *sequences* of steps all of whose *prefixes* are developments of the given set.

[5] Following the rewrite approach, residual systems do *not* assume that steps are closed under composition. Indeed, parallel steps are not, but *reductions* of parallel steps do have compositions and therefore also joins as follows from Proposition 1. In our example, both reductions $\iota(Ix) \cdot \iota(x)$ and $I\iota(x) \cdot \iota(x)$ along the two legs of the diamond are (equivalent) joins of $\iota(Ix)$ and $I\iota(x)$.

[6] For a peak, the choice of valley witnessing (PERM) may be non-deterministic. Essentially this follows since FD makes Newman's Lemma apply 'locally' to developments, allowing to show that independently of the choice the induced redex-trace relation is the same; see the proof of Lem. 3 and cf. [15, Prop. 2.4.16] and [17, Thm. 2].

extensions to $\to$-reductions respectively $\multimap\!\!\to$-reductions) are denoted by overlining respectively underlining, but we omit them as much as possible. Note $\gamma = \underline{(\overline{\gamma})}$ for any γ.

Remark 4. *(FD) entails the equivalence closures of $\to, \multimap\!\!\to$ are the same, but their reflexive–transitive closures may differ if (SD) does not hold: for steps $\phi : a \to b$ and $\phi' : b \to c$ with only $\phi \, [\![\phi]\!] \, \phi'$ non-empty, we have $a \to b$ and indeed also $a \multimap\!\!\to c \leftarrow\!\!\multimap b$, but not $a \multimap\!\!\twoheadrightarrow b$.*

Lemma 3 (SD,FD,PERM)**.** *$\equiv = \equiv_l$ and $\sqsubseteq = \sqsubseteq_l$ (after embedding; in both directions).*

The main result on multi-redexes of [13] is now a matter of chaining the above results:

Proof of Thm. 2. Lem. 2 for $\to$ induces a residual system on $\multimap\!\!\to$. By Prop. 1 that induces a residual system with composition on $\multimap\!\!\twoheadrightarrow$, which by Thm. 1 has lubs and left-cancellation up to *square* homotopy. Hence $\twoheadrightarrow$ has lubs and left-cancellation up to *local* homotopy by Lem. 3. □

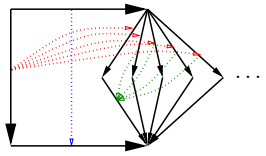


Figure 1: Rewrite system satisfying (SD), (FD) and (PERM) but not (F)

Fig. 1 illustrates the result for a system for which (F) does not hold, a second indication the properties in [13] are too strong. To recover Thms. 1 and 2 of [13] *exactly*, using (F), it suffices to observe that the above can be relativised to a collection $\mathcal{R}$ of sets of redexes such that $\varnothing, \{\phi\} \in \mathcal{R}$ for all redexes ϕ, and $\Phi \cup \Psi, (\Phi \, [\![\Psi]\!]) \in \mathcal{R}$ for all co-initial $\Phi, \Psi \in \mathcal{R}$, and note that the *finite* sets of co-initial steps constitute such a collection. The example in Fig. 1 is rather artificially infinite, but note that although the notion of multi-redex extends naturally (under some provisos) and are at the basis of *infinitary* confluence [20, Ch. 12], (FD) fails for them, a third indication the properties in [13] are too strong.

We generalise *redexes* to *treks* [13], employing $\mathfrak{t}, \mathfrak{s}, \ldots$ ($\mathfrak{T}, \mathfrak{S}, \ldots$) to range over (sets of) them.

Definition 6 ([13])**.** *A* trek*-trace relation maps each step $\phi : a \to b$ to a relation $[\![\phi]\!]$ between the treks of a and b, where (*multi-*)*treks *of a are elements (subsets) of a set $T(a)$ quasi-ordered by $\leq_a$ having the redexes of a as its minimal elements, and such that $\geq_a \cdot [\![\phi]\!] \subseteq [\![\phi]\!] \cdot \geq_b$.*

Intuitively, a trek is a representation of a reduction and $\leq$ a *causal* order on the redexes contracted; the condition $\geq_a \cdot [\![\phi]\!] \subseteq [\![\phi]\!] \cdot \geq_b$ then captures that if a redex has a residual so do the redexes it causes. Accordingly, we restrict $\phi^{\mathfrak{T}}$ in $[\![\to]\!]$ (Def. 4) to steps ϕ in the $\leq$*-downward closure* of $\mathfrak{T}$. After these changes and replacing *redex* by *trek* everywhere[7], *everything* above carries over verbatim, in particular Def. 4, Rem. 2, Lem. 2, Rem. 3, Def. 5, Rem. 4, Lem. 3, the main result Thm. 2, and their proofs, using the following remark in the proof of Lem. 2:

Remark 5. *The properties of $\leq$ make $[\![\to]\!]$ a labelling of itself: if $\phi^{\mathfrak{T}}$ is a $[\![\to]\!]$-step and $\mathfrak{T} \leq \mathfrak{T}'$, i.e. $\mathfrak{t} \leq \mathfrak{T}'$ for all $\mathfrak{t} \in \mathfrak{T}'$, then $\phi^{\mathfrak{T}'}$ is a $[\![\to]\!]$-step by transitivity of $\leq$ (if only $\mathfrak{T} \subseteq \mathfrak{T}'$ then transitivity of $\leq$ is not needed) and $(\mathfrak{T} \, [\![\phi]\!]) \leq (\mathfrak{T}' \, [\![\phi]\!])$ by $\geq \cdot [\![\cdot]\!] \subseteq [\![\cdot]\!] \cdot \geq$ and $[\![\cdot]\!]$ being defined pointwise.*

[7] And also $\in$ into $\leq$ when appropriate, and references to [13, Sect. 2] into corresponding ones to [13, Sect. 3].

Thus we have shown that the axiomatisation of [13] is sufficient but not necessary for obtaining a good residual theory. Although one often *may* factor residual theory through these axioms, there usually is no need to do so, and residual systems can be constructed directly and inductively [8, 20]. We conclude with two remarks on the (FD) axiom:

(FD) was not included among the axioms of residual systems [20] as we did not see a motivation for it. More generally, it is an open question whether finiteness or termination axioms have a place in analysing causality, cf. [21]. Of course, since they give rise to induction measures, they may be practically useful, and we are indeed happy to use them if and when available. For instance, in [15] we showed (FD) to be a consequence of *termination* of the so-called *substitution calculus* (SC) [19] underlying a rewrite format. But for infinitary rewrite systems termination of the SC and hence (FD) are surely too strong, despite that infinitary confluence of orthogonal systems is still based on causality/multi-redexes (up to some provisos).

(FD) may be hard to attain. The application of multi-treks to deal with Lévy's extraction theory for the $\lambda\beta$-calculus in [13, Section 6] is beautiful,[8] but in that application (FD) boils down [13, p. 46] to finiteness of family developments (FFD), cf. [16]. (FFD) is a key result in term rewriting at the basis of standardisation, (hyper-)normalisation of strategies, the theory of optimality, and more, but it also is subtle: It was formulated for the $\lambda\beta$-calculus by Lévy, forming the basis of his beautiful theory of optimality [10], but he resorted [5] to asking the Dutch, van Daalen (whose proof is employed in [10, Sect. II.1.5]) and de Vrijer [4, Stellingen], to prove it.[9] Melliès showed [12, Section 6.2.2] the result [9, Thm. 6.2.4] underlying the proof of (FFD) for Klop's combinatory reduction systems (CRSs) to be incorrect, leaving it and its consequences such as standardisation in limbo. We proved (FFD) for HRSs, hence CRSs, by adapting van Daalen's nifty proof [16], cf. [3].[10]

Acknowledgments Thanks to the IWC 2021 reviewers for feedback.

References

[1] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 2nd revised edition, 1984.

[2] H.J.S. Bruggink. Residuals in higher-order rewriting. In R. Nieuwenhuis, editor, *Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9–11, 2003, Proceedings*, volume 2706 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2003. doi:10.1007/3-540-44881-0_10.

[3] H.J.S. Bruggink. *Equivalence of Reductions in Higher-Order Rewriting*. PhD thesis, Utrecht University, 2008. url:http://dspace.library.uu.nl/handle/1874/27575.

[4] R.C. de Vrijer. *Surjective Pairing and Strong Normalization: Two Themes in Lambda Calculus*. PhD thesis, Universiteit van Amsterdam, January 1987.

[5] R.C. de Vrijer. Personal communication, 1997.

[6] Y. Guiraud, P. Malbos, and S. Mimram. A Homotopical Completion Procedure with Applications to Coherence of Monoids. In F. van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications (RTA 2013)*, volume 21 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 223–238, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.RTA.2013.223.

[8] It seems worthwhile to adapt it to structured rewrite systems such as TRSs, HRSs, and GRSs.

[9] For first-order term rewrite systems (FFD) is due to Maranget [11]; then it is a simple consequence of RPO.

[10] In view of the subtleties it seems of interest to formalise a proof of (FFD) for HRSs in some proof assistant.

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

[8] G. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems, Part I + II. In J.L. Lassez and G.D. Plotkin, editors, *Computational Logic – Essays in Honor of Alan Robinson*, pages 395–443, Cambridge MA, 1991. MIT Press. Update of: Call-by-need computations in non-ambiguous linear term rewriting systems, 1979.

[9] J.W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. Mathematisch Centrum, Amsterdam, 1980.

[10] J.-J. Lévy. *Réductions correctes et optimales dans le λ-calcul*. Thèse de doctorat d'état, Université Paris VII, 1978. url:http://pauillac.inria.fr/ levy/pubs/78phd.pdf.

[11] L. Maranget. *La stratégie paresseuse*. Thèse de doctorat, Université Paris 7, 1992.

[12] P.-A. Melliès. *Description Abstraite des Systèmes de Réécriture*. Thèse de doctorat, Université Paris VII, December 1996.

[13] P.-A. Melliès. Axiomatic rewriting theory VI residual theory revisited. In S Tison, editor, *Rewriting Techniques and Applications, 13th International Conference, RTA 2002, Copenhagen, Denmark, July 22–24, 2002, Proceedings*, volume 2378 of *Lecture Notes in Computer Science*, pages 24–50. Springer, 2002. doi:10.1007/3-540-45610-4_4.

[14] M.H.A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43:223–243, 1942. doi:10.2307/2269299.

[15] V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit, Amsterdam, March 1994. https://research.vu.nl/en/publications/confluence-for-abstract-and-higher-order-rewriting.

[16] V. van Oostrom. Finite family developments. In H. Comon, editor, *Rewriting Techniques and Applications, 8th International Conference, RTA-97, Sitges, Spain, June 2-5, 1997, Proceedings*, volume 1232 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 1997. doi:10.1007/3-540-62950-5_80.

[17] V. van Oostrom. Confluence by decreasing diagrams; converted. In A. Voronkov, editor, *Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings*, volume 5117 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2008. doi:10.1007/978-3-540-70590-1_21.

[18] V. van Oostrom. Multi-redexes and multi-treks induce residual systems; least upper bounds and left-cancellation up to homotopy. Long version. url:http://cl-informatik.uibk.ac.at/users/vincent/research/publication/pdf/axrs-iwc-long-2021.pdf, June 2021.

[19] V. van Oostrom and F. van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. In J. Heering, K. Meinke, B. Möller, and T. Nipkow, editors, *Higher-Order Algebra, Logic, and Term Rewriting, First International Workshop, HOA '93, Amsterdam, The Netherlands, September 23–24, 1993, Selected Papers*, volume 816 of *Lecture Notes in Computer Science*, pages 276–304. Springer, 1993. doi:10.1007/3-540-58233-9_13.

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

[21] G. Winskel. An introduction to event structures. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 364–397, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg. doi:10.1007/BFb0013026.

Wrap ambiguities and how to enumerate them

Lars Hellström[1]

Division of Applied Mathematics and Physics, The School of Education, Culture and Communication, Mälardalen University, Box 883, 721 23 Västerås, Sweden
lars.hellstrom@mdh.se

Abstract

Network rewriting can be summarised as a generalisation of term rewriting to support that operations can have multiple out-parameters (coarity greater than 1) as well as the traditional multiple in-parameters (arity greater than 1). When fleshing out this idea, one is forced to make certain choices, which are discussed in this paper; network rewriting represent one way of making these choices: respect algebraic linearity, preserve acyclicity, and stay abstract (as opposed to imposing a geometric foundation).

In a 2014 IWC paper, it was reported that for network rewriting there emerges a third kind of ambiguity (critical pair) besides the classical overlap and inclusion ambiguities, namely wrap ambiguities where the way two redexes wrap around each other without overlapping can cause a rewrite of one to block the other. At that point it was not known how to enumerate these ambiguities, but here a generic method for this based on boolean matrices and SAT-solving is presented.

1 Discussion of models

Term rewriting operates on expressions as formalised in mathematical logic, where every combination of subexpressions to make a larger expression is by the use of an abstract function symbol taking zero or more arguments, and every expression is either a function application or a variable. However in modern algebra it is increasingly becoming necessary to deal with expressions that do not easily fit into this model; these theories comprise operations that vary not only in the number of in-parameters (arguments) they take, but also in the number of out-parameters they produce. To deal with these natively, one may relax the unspoken constraint that every expression has an underlying rooted (hence directed) tree structure, to allow the underlying structure to be that of a directed graph: operations are still vertices, there is an incoming edge for every in-parameter, and an outgoing edge for every out-parameter; an edge from one vertex to another means that an out-parameter of the first vertex is identified with an in-parameter of the second. Expressions are thus modelled as something like data-flow *networks*.

Whereas it may seem obvious that the optimal implementation of a certain computation may well be in terms of subroutines with multiple out-parameters—for example a single division operation that returns both a quotient and a remainder—it need not be immediately clear why said computation could not specified in terms of only single-result operations (such as separate quotient and remainder). A full explanation of this would have to explore the differences between cartesian and tensor products, but that is too long a digression to get into here; the interested reader may instead see [1]. The heart of the matter is however that many of the theories which make use of these operations with multiple results depend critically upon these being *entangled*, which means they have to be computed together.

In fact the interpretation of more general graphs as denoting expressions is not a trivial matter; the recursive interpretation for terms depends critically on them having a tree structure, which we just rejected. A data-driven evaluation—determine values on the outgoing edges from

a vertex once values have been determined for all its incoming edges—is not out of the question, but would need to deal with entanglement explicitly, thus losing in generality. Instead the interpretation is mostly by decomposing the directed graphs into elementary pieces for which (function) values are given, and then a suitable algebraic structure (often a category) is used to recompose these elementary values into a value for the whole. The means of composition that this algebraic structure provides places restrictions on what a graph may look like if it is to be interpreted as an expression.

One axis of variation is what topological structure (if any) these graphs should be embedded into, which corresponds to the choice between symmetric, braided, or plain monoidal categories for governing the recomposition process. The symmetric case corresponds to abstract graphs, and should thus be the natural choice from a computer science or logic point of view (being less of an ontological commitment), but the plain and braided cases appear to be more popular in the category theory literature. A reason for that popularity is likely that topology has long been a prominent area for applications of category theory.

Another axis of variation is whether cycles should be allowed, and if so how an interpretation of those is concretely achieved; cycles in a data-flow network naively cause deadlock, when an operation vertex is waiting for input that (possibly in several steps) depend on an output of said vertex. Classically terms may be given something like a cycle in the underlying graph structure through some manner of fixed-point operator, but the rewriting of such is not entirely trivial. On the category side, the most direct way of supporting cycles is through the use of *traced* monoidal categories, where there is an operation trace/contraction/feedback that allows identifying an output with an input. A less direct way is by introducing 'cap' and 'cup' operations satisfying the zig-zag identities—this can be done as a matter of rewriting, but is often worked into the notation as 'raising/lowering indices' or 'bending edges' (allowing both endpoints to be heads or both tails)—and in particular the caps make heavy use of entanglement. However traces, caps, and cups can all be problematic when it comes to their interpretation in concrete applications; as a rule of thumb they are straightforward in finite-dimensional cases (the trace of matrix is trivial to evaluate) but may be impossible in infinite-dimensional cases (the trace of the identity operator becomes infinite). Hence it is for a general rewriting framework safest disallow cycles in expression, leaving it to users to add explicit caps and cups where appropriate.

A third axis concerns whether multiple edges may attach to the same "port" of a vertex, or equivalently, whether (internal) edges should have exactly two endpoints. Term graphs certainly suggest that using a single output as input in multiple places has its uses, and the share graphs of Hasegawa [2] aim to support this; likewise Ştefănescu [5] cover a number of variations in this regard, and also give examples of where such things may be appropriate. However in higher algebra there are strong reasons not to allow such things—changing the multiplicity of an intermediate result completely destroys multilinearity. In practice it is straightforward to introduce explicit operations for duplicating (coproduct) or destroying (counit) data, so a 1-to-1 restriction on the graphs is not a significant expressive loss, and several interesting algebraic theories even arise as a reformulation of classical theories to satisfy this linearity constraint; Hopf algebras arise from groups in that way. An notable consequence for rewriting is that unification disappears as a separate problem; it rather happens implicitly as part of overlaps involving rules for the coproduct and counit.

Finally it may be remarked that the 'monoid' in 'monoidal category' refers to the fact that the set of types supported constitute a monoid under concatenation/tensor product. In practice only free monoids seem to be used, which corresponds to having a set of atomic types given by the user, that do not interact except in operation vertices. A graph model for this should then make sure to label every edge with one of these atomic types—the type of data that

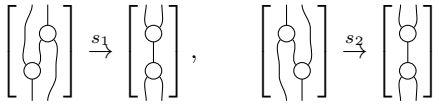
may be carried along that edge—but for rewriting that is mostly rendundant since the type of an edge can be inferred from the vertices it is incident with, and overlapping existing type-consistent graphs will only generate new type-consistent graphs. Hence it is perfectly possible (an notationally easier) to set up the rewriting framework as being untyped, in which case symocats simplify to what MacLane called a PROP. It turns out *networks*—directed acyclic open graphs where vertices are with respect to in- and out-degree consistently decorated with symbols from a doubly ranked alphabet and each edge attach to a separate port of a vertex it is incident with—are modulo network isomorphism exactly the elements of the free PROP generated by that doubly ranked alphabet [3, Sec. 5].

2 Formal feedbacks

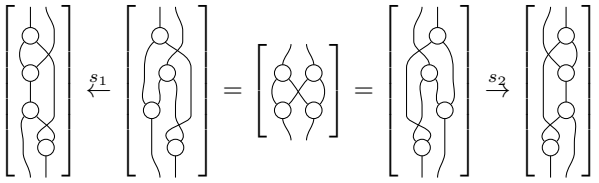
Even if cycles can be problematic for the interpretation of networks as expressions, they are quite useful when it comes to analysing the structure of networks, since they permit expressing any whole as an A part beside a B part, having some outputs of that $A \otimes B$ combinations connected back to select inputs of it, without getting into details of whether A comes before B dependency-wise, B comes before A, or in fact it might be both. Interestingly enough, this is possible even in the free PROP, since it supports *formal feedbacks* [3, Sec. 9].

The idea is that one may to any network G associate a boolean matrix $\mathrm{Trf}(G)$ called the *transferrence* of G: this has a 1 in position (i, j) iff there is a directed path in G from input j to output i; this Trf may also be interpreted as a PROP homomorphism into the PROP of boolean matrices. If G and H are networks whose transferrences have block matrix decompositions $\mathrm{Trf}(G) = \left[\begin{smallmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{smallmatrix}\right]$ and $\mathrm{Trf}(H) = \left[\begin{smallmatrix} b_{22} & b_{23} \\ b_{32} & b_{33} \end{smallmatrix}\right]$ where a_{22} is $q \times r$ and b_{22} is $r \times q$, then the matrix $a_{22}b_{22}$ is nilpotent iff the *symmetric join* $G \bowtie_r^q H$ is acyclic, that one obtains by identifying the q last outputs of G with the q first inputs of H and likewise the r first outputs of H with the r last inputs of G. This carries over to the free PROP, which canonically comes with a filtration $\mathcal{F}$ indexed by boolean matrices, such that $\mathcal{F}_a$ is the set of all elements μ of the free PROP that have a transferrence $\leq a$ in the standard matrix order. $\mathcal{F}$ being a PROP filtration, it follows from $\mu \in \mathcal{F}_a$ and $\nu \in \mathcal{F}_b$ that $\mu \circ \nu \in \mathcal{F}_{a \circ b}$ (if $a \circ b$ is defined) and $\mu \otimes \nu \in \mathcal{F}_{a \otimes b}$, but more importantly each $\bowtie_r^q$ may, provided a and b satisfy the above nilpotency condition, be regarded as an operation $\mathcal{F}_a \times \mathcal{F}_b \longrightarrow \mathcal{F}_c$ for $c = \left[\begin{smallmatrix} a_{11}+a_{12}b_{22}(a_{22}b_{22})^*a_{21} & a_{12}(b_{22}a_{22})^*b_{23} \\ b_{32}(a_{22}b_{22})^*a_{21} & b_{33}+b_{32}a_{22}(b_{22}a_{22})^*b_{23} \end{smallmatrix}\right]$, where $p^* = \sum_{k=0}^{\infty} p^k$ denotes the *Kleene star* of the boolean matrix p. Joining with a width q identify is also known as the width q *(formal) feedback* $\uparrow^q$. A symmetric join on all inputs and outputs of the right factor is for simplicity denoted $\rtimes$.

In [3, Sec. 10] this was used to construct a rewriting theory for networks, where a rewrite rule $\mu \to \mu'$ with $\mu, \mu' \in \mathcal{F}_b$ could be placed into a context defined by some $\nu \in \mathcal{F}_a$ and used to do $\nu \rtimes \mu \to \nu \rtimes \mu'$ whenever that symmetric join is defined; the machinery of formal feedbacks make it feasible to ensure that this always respects the ordering with which these rules are compatible. It was however in the detailed analysis of the resulting ambiguities (critical pairs) observed that one could not claim that resolving only overlap and inclusion ambiguities would suffice unless assuming that $b = \mathrm{Trf}(\mu_1)$ (rewrite rule is "sharp"), and in [3, Ex. 10.28] an example was given of a *wrap ambiguity* where two redexes would block each other despite being disjoint, by virtue of each having a input that depends on output from the other. The two rules



give rise to the critical pair

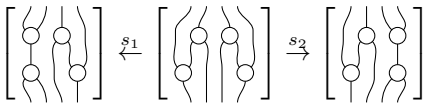


(1)

Characterising those situations where this happened were then left as an open problem. A compounding factor is that this mainly seemed to arise where the transferrence of a network would change without decreasing (rather increase or be outright incomparable), which is troublesome when one seeks to find a compatible ordering: the rules involved in this are often also not (easily) orientable. That is however a different story.

3 Networks with obligations

Eliding the interconnections between the two redexes, which would go from output 3 to input 1 and from output 2 to input 4, the ambiguity looks like



In either branch, applying the other rule to its redex is blocked because doing so would create a cycle due to the change introduced by the first rule, whereas in the initial configuration both rules can be applied. Since nilpotency of transferrence matrices allows us to test for acyclicity, we can formulate a condition for the transferrence $\left[\begin{smallmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{smallmatrix}\right]$ of a context ν that would cause this: the block p_{22} connecting outputs of the ambiguity back to inputs thereof must have

$$\begin{pmatrix} 1&1&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&1&1 \end{pmatrix} p_{22} \text{ nilpotent}, \qquad \text{but } \begin{pmatrix} 1&1&0&0 \\ 1&1&0&0 \\ 0&0&1&0 \\ 0&0&1&1 \end{pmatrix} p_{22}, \begin{pmatrix} 1&1&0&0 \\ 0&1&0&0 \\ 0&0&1&1 \\ 0&0&1&1 \end{pmatrix} p_{22} \text{ non-nilpotent.} \tag{2}$$

The only solution to this is that $p_{22} = \begin{pmatrix} 0&0&1&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&1&0&0 \end{pmatrix}$, which indeed places us in the situation depicted in (1).

How could one automatically solve such problems, nilpotency being a rather nonlinear constraint? It turns out that they can quite conveniently be formulated as boolean satisfiability problems, with the elements of p_{22} as individual boolean variables. Nilpotency as such may seem difficult to encode, but the nilpotency of a boolean $n \times n$ matrix A is equivalent to the claim that $A^n = 0$, and repeated boolean multiplications are straightforward to encode if one introduces helper variables for the elements of the intermediate products; exponentiation by squaring helps to further reduce the number of multiplications that need to be encoded. Non-nilpotency of a boolean matrix A is conversely equivalent to the claim that its Kleene plus $A^+ = AA^* = \sum_{k=1}^{n} A^k$ does not have a zero diagonal, which again is thus possible to encode in terms of repeated boolean multiplications.

In hindsight, a problem with the [3] rewriting theory is that it assigns a transferrence to each rule, when what it in fact needs is to know what restrictions may be imposed by the context in which the rule is to act. For the derived rule a completion would produce from (2) the least

possible transferrence is $\left(\begin{smallmatrix} 1&1&0&0\\ 1&1&0&0\\ 0&0&1&1\\ 0&0&1&1 \end{smallmatrix}\right)$, but that is also too large to allow that both dependencies expressed by the p_{22} matrix above—this derived rule would not be able to resolve the very ambiguity from which it was derived! That is clearly not satisfactory, but such is sometimes the lure of the algebra; the theory of the PROP filtration $\mathcal{F}$ simply looked too good for it to not be the right basis for the rewrite theory. It still has its uses, but it is not what should characterise the rewrite rules.

A better approach is instead to let each rewrite rule come with an *obligation* of supporting a certain amount of feedback imposed by the context in which it is to operate—essentially that p_{22} matrix derived above. The basic sets $\mathcal{Y}(r)$ of objects being rewritten are indexed by boolean matrices r, and consist of all μ in the free PROP such that $\mathrm{Trf}(\mu)r$ is (defined, square, and) nilpotent; ordinary algebraic expressions have obligation $r = 0$, but higher obligations arise when resolving ambiguities. A rule $\mu \to \mu'$ supporting obligations r can be applied to make the rewrite step $\nu \rtimes \mu \to \nu \rtimes \mu'$ while respecting obligations q iff the transferrence $\mathrm{Trf}(\nu) = \left[\begin{smallmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{smallmatrix}\right]$ also satisfies (i) that qp_{11} is nilpotent and (ii) that $p_{21}q(p_{11}q)^*p_{12} + p_{22} \leqslant r$; in other words the context does not by ifself violate the target obligations q, and combining the context with those obligations does not create effective obligations exceeding those that this rule supports.

For enumerating wrap ambiguities, this leads to a slightly more complicated set of constraints that just the (2) combination of nilpotency and non-nilpotency, but it is all possible to handle with the same set encoding tricks, of which the theoretically foremost is that one need only consider matrix powers up to a known bound.

References

[1] John C. Baez and Mike Stay. Physics, Topology, Logic and Computation: A Rosetta Stone. Pp. 95–174 in *New Structures for Physics* (ed. Bob Coecke), Lecture Notes in Physics vol. **813**, Springer, Berlin, 2011. arXiv:0903.0340v3.

[2] Masahito Hasegawa. *Models of sharing graphs.* CPHC/BCS Distinguished Dissertations. Springer-Verlag London, Ltd., London, 1999. A categorical semantics of let and letrec, Dissertation, University of Edinburgh, Edinburgh. doi:10.1007/978-1-4471-0865-8.

[3] Lars Hellström. Network Rewriting I: The Foundation, April 2012. arXiv:1204.2421 [math.RA]. arXiv:1204.2421.

[4] André Joyal and Ross Street. The Geometry of Tensor Calculus, I. *Adv. Math.* **88** (1991), 55–112.

[5] Gheorghe Ştefănescu. *Network Algebra.* Springer, 2000. ISBN 1-85233-195-X.

Completion of operadic rewriting systems by Gaussian elimination

Benjamin Dupont[1], Philippe Malbos[2], and Isaac Ren[3]

[1,2] Université de Lyon, France, {bdupont,malbos}@math.univ-lyon1.fr
[3] École Normale Supérieure de Lyon, France, isaac.ren@ens-lyon.fr

Abstract

We study the confluence properties of non-symmetric operadic rewriting systems using linear algebra methods. We extend the completion procedure F4, known for commutative and non-commutative algebras, to operads. This procedure allows us to parallelize completion by applying a Gaussian elimination process in order to treat multiple critical branchings simultaneously. We discuss heuristics and strategies to optimize this procedure in the operadic context: first to reduce the set of critical branchings to be examined and then to parallelize the elimination.

1 Introduction

Algebraic rewriting theory aims at studying rewriting relations in algebraic and categorical structures such as monoids, categories, equational theories, linear algebras, operads and higher-dimensional algebras, and categories. Proofs of confluence of an algebraic rewriting system (AlgRS) are mainly based on the critical branching lemma (CBL) that proves local confluence from confluence of a set of *critical branchings*, which correspond to confluence obstructions induced by minimal overlappings of rules. The CBL approach is used various contexts, including automated theorem proofs, word problems in universal algebras, and polynomial ideal membership. CBL's were proved for numerous AlgRS's: rewriting on strings [24], terms [18], and higher-dimensional categories [14, 15]. CBL's also have various formulations in linear structures, for commutative algebras [4], associative algebras [1, 2, 23], non-symmetric and shuffle operads [7, 20], and higher-dimensional linear categories [10]. In algebraic rewriting, the CBL constitutes the first step in the construction of cofibrant replacements of algebraic and categorical structures [13, 15, 19].

The principle of the critical branching completion procedure (CBCP) on an AlgRS can be formulated as follows:

Input: A set R of rules of an algebraic rewriting system.

$\mathsf{R}' := \mathsf{R}$;
$\mathcal{C} :=$ critical branchings of R';
while $\mathcal{C} \neq \varnothing$ **do**
  Select a subset B of branchings in $\mathcal{C}$, and remove them from $\mathcal{C}$;
  Add rewriting rules to R' to make the non-confluent branchings of B confluent;
  Update $\mathcal{C}$ with branchings induced by the new rules;
return R';

If the additional rewriting rules are oriented with respect to a termination order, such as a monomial order, the procedure returns a terminating rewriting system. If moreover the procedure terminates, then the result is a convergent rewriting system. Otherwise, it builds an increasing sequence of rewriting systems, whose limit is convergent. The resulting rewriting system is finite if and only if the input is finite and the procedure terminates.

Concrete implementations of CBCP are based on the preparation of the AlgRS (autoreductions and adding new generators), the type of critical branchings considered, a filtration on the critical branchings, and the parallelization of the computation of their confluence at each step. The choice of branchings to consider can depend on a study of overlapping patterns (Buchberger's criterion [5], Triangle Lemma [3]), or relations between critical branchings and Gebauer-Möller criteria [12, 20, 22]. The filtration on critical branchings gives the order in which to examine the critical branchings and depends on the shape of the rules (reduced, homogeneous...). Finally, several methods can be used to compute confluence in parallel wrt the filtration. One approach is based on Gaussian elimination, mainly developed for the computation of commutative [11] and non-commutative Gröbner bases [6, 25], see also [16]. This principle also appears in [3] for the study of non-symmetric operads.

However, these optimizations were not fully developed in the case of operadic rewriting. Indeed, this algebraic paradigm is complex due to the linear context, the problem of managing symmetric actions, and the complexity of operadic patterns. Rewriting systems for non-symmetric operads were studied in [3, 8, 20], and the question of management of the action of symmetries on terms was addressed in [7], which introduces a notions of Gröbner bases for shuffle operads, and implemented in [9].

In this work, we study the optimization of completion procedures for operadic rewriting systems (ORS). We define a completion algorithm for ORS resolving non-confluence by Gaussian elimination with respect to a chosen confluence obstruction strategy. This work is part of a general program that aims to define computational tools for mathematicians studying higher algebras and higher categories. Indeed, novel higher structures appear in numerous fields such as geometry, physical mathematics, representation theory and quantum topology. Higher structures are generally defined by complex presentations by generators and relations, so there is real need for efficient completion procedures in algebraic contexts.

This abstract is organised as follows. In Section 2, we recall the notion of rewriting systems for non-symmetric operads and we explain some strategies for the implementation of CBCP. Section 3 presents the completion algorithm for ORS's by Gaussian elimination.

2 Confluence of operadic rewriting systems

In this section we recall the notion of operadic rewriting systems on a ground field $\mathbb{K}$ of zero characteristic and the different approaches to obtaining a CBL for these systems.

2.1. Operadic rewriting systems. A *collection* is a sequence $(V(n))_{n\in\mathbb{N}}$ of vector spaces indexed by *arities* $n \geqslant 0$. A *(non-symmetric) operad* is a collection P with an *identity* element $\varepsilon \in P(1)$, and equipped with composition maps $\circ : P(k)\otimes P(n_1)\otimes\ldots\otimes P(n_k) \to P(n_1+\ldots+n_k)$ satisfying identity and associativity conditions. The set of *monomials* $\mathcal{T}(\Sigma)$ is the term algebra on a graded set $\Sigma = (\Sigma(n))_{n>0}$. As for the free algebra generated by a family of indeterminates, we define the free operad $\mathcal{F}(\Sigma)$ on Σ, where, for $n > 0$, $\mathcal{F}(\Sigma)(n)$ is the vector space spanned by monomials of arity n, called *(homogeneous) polynomials*. The *support* of $f = \sum_{i\in I} \lambda_i u_i$ is the set of monomials $\mathrm{Supp}(f) := \{u_i \,|\, i \in I\}$ that appear in its decomposition. A *context of* $\mathcal{F}(\Sigma)$ *of inner arity* k is a term C of $\mathcal{T}(\Sigma \cup \{\Box_k\})$, where $\Box_k$ is a symbol of arity k that appears exactly once in C. For a monomial u of arity k, we denote by $C[u]$ the monomial C where we replace $\Box_k$ by u; we extend this notation to polynomials by linearity.

An *operadic rewriting system* (ORS) is the data $X = (\Sigma, R)$ made of a graded set Σ and a relation $R \subset \mathcal{T}(\Sigma) \times \mathcal{F}(\Sigma)$, whose elements are *rewriting rules* $\alpha : s(\alpha) \to t(\alpha)$. We define the graph $\mathcal{R}_X$, whose vertices are the elements of $\mathcal{F}(\Sigma)$ and whose edges are the $\lambda C[\alpha] + 1_b$:

$\lambda C[s(\alpha)] + b \to_R \lambda C[t(\alpha)] + b$, where $\alpha \in R$, C is a context, $\lambda \in \mathbb{K} \setminus \{0\}$, and b is a polynomial of $\mathcal{F}(\Sigma)$. An edge of $\mathcal{R}_X$ is a *rewriting monomial* when $\lambda = 1$ and $b = 0$, and a *rewriting step* when $C[s(\alpha)] \notin \mathrm{Supp}(b)$. Denote by $\mathcal{R}_X^m$ the set of rewriting monomials of X and by $\cdot$ the composition of paths in $\mathcal{R}_X$. The paths in $\mathcal{R}_X$ made of rewriting steps are called *rewriting paths of* X. A polynomial a in $\mathcal{F}(\Sigma)$ is in *normal form wrt* X if there is no rewriting step with source a. A *reduction strategy* is a map σ, which to any monomial u associates an identity if u is reduced, and a rewriting monomial $\sigma(u)$ of source u otherwise. The ORS X is *terminating* if there does not exist an infinite rewriting path.

A *monomial order* on $\mathcal{T}(\Sigma)$ is a total order $\prec$ stable by product, that is, for all $u, u' \in \mathcal{T}(\Sigma)(k)$, $v, v' \in \mathcal{T}(\Sigma)(\ell)$, and $1 \leqslant i \leqslant k$, $(u \prec u', v \prec v')$ implies $u \circ_i v \prec u' \circ_i v'$. An ORS X is *compatible with* $\prec$ if, for every rewriting rule $\alpha \in R$ and every monomial $v \in \mathrm{Supp}(t(\alpha))$, $v \prec s(\alpha)$. Note that if $\prec$ is well-founded, then X is terminating.

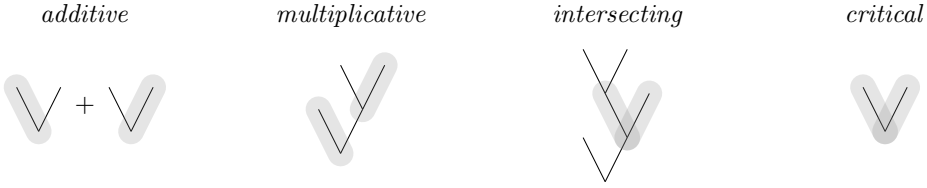
A *branching* (resp. *local branching*) is a pair (f, g) of rewriting paths (resp. rewriting steps) such that $f \neq g$ and $s(f) = s(g)$. The local branchings of X are classified as follows:

i) *additive branchings*: $(\lambda f + \mu 1_v + 1_c, \lambda 1_u + \mu g + 1_c)$, where $f : u \to a, g : v \to b \in \mathcal{R}_X^m$, $\lambda, \mu \in \mathbb{K} \setminus \{0\}$, c is a 0-cell, $u \neq v$, and $u, v \notin \mathrm{Supp}(c)$.

ii) *multiplicative branchings*: $(\lambda C[f, 1_v] + 1_c, \lambda C[1_u, g] + 1_c)$, where C is a two-hole context, $f : u \to a, g : v \to b \in \mathcal{R}_X^m$, $\lambda \in \mathbb{K} \setminus \{0\}$, c is a 0-cell, and $C[u, v] \notin \mathrm{Supp}(c)$.

iii) *intersecting branchings*: the rest of the local branchings. A *critical branching* is an intersecting branching that is minimal for the order induced by $(f, g) \subseteq (C[f] + 1_c, C[g] + 1_c)$ for a context C and a polynomial c of $\mathcal{F}(\Sigma)$.

In a schematic way, we can illustrate local branchings for an ORS as follows, where the highlighted parts of tree monomials indicate the sources of the rewriting rules:



A branching (f, g) is *confluent* if there exist rewriting paths h and k such that $t(f \cdot h) = t(g \cdot k)$. Given a set B of branchings of X, X is B-*confluent* if every $b \in B$ is confluent. If B is the set of all branchings, then we say that X is confluent. We say that X is *convergent* if it is terminating and confluent.

2.2. Strategies for completion procedures. A completion procedure wrt a given monomial order $\prec$ transforms an ORS into a convergent one by adding rules, oriented wrt the order $\prec$, to amend non-confluent branchings. Such a procedure is based on a map that selects a type of branching whose confluence implies the confluence of all branchings, defined as follows.

A map $\mathcal{CO}$ that associates to every ORS X a set of branchings $\mathcal{CO}(X)$ of X is a *confluence obstruction map* when every terminating ORS X is confluent iff it is $\mathcal{CO}(X)$-confluent. For example, there exists a minimal confluence obstruction map $\mathcal{M}$ defined as $\mathcal{M}(X) = \varnothing$ if X is confluent, and $\mathcal{M}(X) = \{b\}$ if X is non-confluent and b is a non-confluent branching. However, it is impracticable to write a completion procedure wrt $\mathcal{M}$, as it would imply being able to determine confluence and compute a non-confluent branching in the first place.

Another approach is to consider confluence-generating sets of branchings. A set B of branchings of an ORS X is *confluence-generating* if, for any branching (f, g) of X, there exist branchings

$(f_1, g_1), \dots, (f_n, g_n)$, which are additive, multiplicative, or in B, rewriting paths f' and g', and contexts $C_1, \dots, C_n$ such that $f = C_1[f_1] \cdot f'$, $g = C_n[g_n] \cdot g'$, and for all $1 \leqslant i \leqslant n-1$, $C_i[g_i] = C_{i+1}[f_{i+1}]$. We get the following lemma:

2.3. Lemma. *A map $\mathcal{B}$ that associates to every ORS X a confluence-generating set of branchings $\mathcal{B}(X)$ is a confluence obstruction map.*

The converse is not true, however: consider $\mathcal{M}(X)$, which is not confluence-generating as soon as X is confluent with a branching.

There are several examples confluence-generating sets in the literature. The classical one is the set of critical branchings, in which case Lemma 2.3 is the CBL, also called Buchberger's criterion for linear rewriting systems [4, 23]. Smaller confluence-generating sets were developed to take into account additional relations between critical branchings. These sets, along with the corresponding proofs of Lemma 2.3, were defined for commutative algebras [5, 22], non-commutative algebras [16, 17], and non-symmetric operads [20].

Small confluence-generating sets appear to be a good compromise between minimizing the size of a confluence obstruction map and minimizing the number of times the confluence obstruction map is called. The question is then to find a minimal confluence-generating set. In certain cases, the answer is known: for instance, for quadratic ORS's, critical branchings form a minimal confluence-generating set.

3 Confluence by elimination

Linear rewriting can be done without a monomial order [13, 19], but in most applications the rewriting rules are compatible with a monomial order. In this case convergent AlgRS's are *Gröbner bases*, and rewriting properties are formulated algebraically. Branchings are described by *S-polynomials* and confluence means that every S-polynomial reduces to zero. Finally, the elimination of critical branchings is encoded by relations among relations (syzygies). In this section we give an implementation of the CBCP for ORS using Gaussian elimination inspired by the F4 algorithm [16].

Fix an ORS $X = (\Sigma, \prec, R)$ compatible with a monomial order $\prec$. Let $P = \{f_1, \dots, f_n\}$ be a set of rewriting monomials on Σ, and consider the totally ordered set $\mathrm{Supp}(P) := \cup_{f \in P} \mathrm{Supp}(s(f) - t(f)) = \{u_1 \prec \cdots \prec u_k\}$. We define the matrix $M_P \in \mathcal{M}_{n,k}(\mathbb{K})$ where $(M_P)_{i,j}$ is the coefficient of u_j in $s(p_i) - t(p_i)$. Thus we can read the elements of P as the rows of M_P, where the largest nonzero coefficient is the source monomial and the other coefficients correspond to the target polynomial. For examples, see the matrices in the appendix.

The first step of completion is as follows. We fix a reduction strategy σ. For each rewriting monomial p of P, we calculate a reduction path, starting with p, from $s(p)$ to a normal form, which follows σ after the first step. We then re-

```
GetRM(σ)(X, P)
Input: An ORS X = (Σ, ≺, R),
A list of rewriting monomials P.
Output: A list of rewriting monomials R'.
R' := P;
T := ∪_{f∈P} Supp(t(f));
treated := lm(P);
while T ≠ ∅ do
    select u ∈ T;
    T := T \ {u};
    treated := treated ∪ {u};
    if σ(u) not an identity then
        R' := R' ∪ {σ(u)};
        T := T ∪ {Supp(t(σ(u))) \ treated};
return R';
```

turn the set $\mathsf{R}' := \mathsf{GetRM}(\sigma)(\mathsf{X}, \mathsf{P})$ of rewriting monomials wrt R that appear in these paths. As for the case of non-commutative algebras [16, Prop. 4.21], if P is finite, then $\mathsf{GetRM}(\mathsf{X}, \mathsf{P})$ terminates.

The next step is to reduce the matrix $\mathsf{M}_{\mathsf{R}'}$ to its row reduced echelon form, $\mathsf{RowReduce}(\mathsf{M}_{\mathsf{R}'})$, by Gaussian elimination. The resulting rows whose largest monomials are not sources of rewriting monomials in R' form a set of new rewriting rules P', which is the result of $\mathsf{Reduction}(\mathsf{X}, \mathsf{P})$.

$\mathsf{Reduction}(\mathsf{X}, \mathsf{P})$
Input: An ORS $\mathsf{X} = (\Sigma, \prec, \mathsf{R})$,
A list of rewriting monomials P.
Output: A list of rewriting rules P'.

$\mathsf{R}' := \mathsf{GetRM}(\sigma)(\mathsf{X}, \mathsf{P})$;
$\mathsf{M}' := \mathsf{RowReduce}(\mathsf{M}_{\mathsf{R}'})$;
$\mathsf{P}' := \{\alpha$ row of M'
  $\mid \alpha \neq 0$ and $\mathsf{s}(\alpha) \notin \mathsf{s}(\mathsf{R}')\}$;
return P';

Finally, we choose a confluence obstruction map $\mathcal{CO}$ and a *selection strategy* $\mathcal{S}$, that returns a subset of branchings, in order to parallelize the completion procedure. The selection strategy in the procedure $\mathsf{F4}$ is equivalently a filtration on `Branchings`. For instance, the *normal selection strategy* consists in filtering branchings by weight of the source, and starting with those of minimal leading weight [11]. For homogeneous presentations, this appears to works well.

3.1. Theorem. *Let X be an ORS, $\mathcal{CO}$ a confluence obstruction map and $\mathcal{S}$ a selection strategy. If the procedure $\mathsf{F4}(\mathcal{C}, \mathcal{S})$ terminates on X, then the ORS $\mathsf{F4}(\mathcal{CO}, \mathcal{S})(\mathsf{X})$ is convergent.*

The proof works as follows. $\mathsf{F4}(\mathcal{CO}, \mathcal{S})(\mathsf{X})$ terminates only if, at some iteration of the first while loop, P' is an empty set for every iteration of the second while loop. This only happens if X' is $\mathcal{CO}(\mathsf{X}')$-convergent, which is equivalent to convergence of X'.

Note that an associative algebra can be seen as a non-symmetric operad concentrated in arity 1. By specifying $\mathcal{CO}$ and $\mathcal{S}$ and restricting $\mathsf{F4}$ to associative algebras, we recover some previously published procedures. If $\mathcal{CO}$ returns the set of critical branchings, we get the non-commutative F4 procedure introduced in [25]. If $\mathcal{S}$ selects a single branching and $\mathcal{CO}$ returns the set of critical branchings, we get the non-commutative Buchberger procedure [1, 2]. If $\mathcal{CO}$ eliminates critical branchings following the optimizations of [17, 25] (interreduction and chain criterion), then we recover their procedures.

$\mathsf{F4}(\mathcal{CO}, \mathcal{S})(\mathsf{X})$
Input: An ORS $\mathsf{X} = (\Sigma, \prec, \mathsf{R})$.
Output: A convergent ORS
  $\mathsf{X}' = (\Sigma, \prec, \mathsf{R}')$.

$\mathsf{R}' := \mathsf{R}$;
AddedRules := true;
while AddedRules **do**
  AddedRules := false;
  Branchings := $\mathcal{CO}(\Sigma, \mathsf{R}')$;
  while Branchings $\neq \varnothing$ **do**
    $\mathsf{B} :=$ $\mathcal{S}$(Branchings);
    Branchings := Branchings $\setminus$ B;
    $\mathsf{P} := \cup_{\{\mathsf{f},\mathsf{g}\}\in\mathsf{B}}\{\mathsf{f}, \mathsf{g}\}$;
    $\mathsf{P}' := \mathsf{Reduction}((\Sigma, \prec, \mathsf{R}'), \mathsf{P})$;
    if $\mathsf{P}' \neq \varnothing$ **then**
      $\mathsf{R}' := \mathsf{R}' \cup \mathsf{P}'$;
      AddedRules := true;
return $(\Sigma, \prec, \mathsf{R}')$;

References

[1] George M. Bergman. The diamond lemma for ring theory. *Adv. in Math.*, 29(2):178–218, 1978.

[2] Leonid A. Bokut. Imbeddings into simple associative algebras. *Algebra i Logika*, 15(2):117–142, 245, 1976.

[3] Murray R. Bremner and Vladimir Dotsenko. *Algebraic operads*. CRC Press, Boca Raton, FL, 2016. An algorithmic companion.

[4] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal (An Algorithm for Finding the Basis Elements in the Residue Class Ring Modulo a Zero Dimensional Polynomial Ideal)*. PhD thesis, Mathematical Institute, University of Innsbruck, Austria, 1965. English translation in J. of Symbolic Computation, Special Issue on Logic, Mathematics, and Computer Science: Interactions. Vol. 41, Number 3-4, Pages 475–511, 2006.

[5] Bruno Buchberger. A criterion for detecting unnecessary reductions in the construction of gröbner bases. In *Symbolic and Algebraic Computation*, pages 3–21, 01 1979.

[6] Cyrille Chenavier. A lattice formulation of the noncommutative F_4 procedure. *Internat. J. Algebra Comput.*, 29(1):23–40, 2019.

[7] Vladimir Dotsenko and Anton Khoroshkin. Gröbner bases for operads. *Duke Math. J.*, 153(2):363–396, 2010.

[8] Vladimir Dotsenko and Bruno Vallette. Higher Koszul duality for associative algebras. *Glasg. Math. J.*, 55(A):55–74, 2013.

[9] Vladimir Dotsenko and Mikael Vejdemo-Johansson. Implementing Gröbner bases for operads. In *OPERADS 2009*, volume 26 of *Sémin. Congr.*, pages 77–98. Soc. Math. France, Paris, 2013.

[10] Benjamin Dupont. Rewriting modulo isotopies in pivotal linear (2,2)-categories. submitted preprint, arXiv:1906.03904, June 2019.

[11] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, June 1999.

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

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

[14] Yves Guiraud and Philippe Malbos. Higher-dimensional categories with finite derivation type. *Theory Appl. Categ.*, 22:No. 18, 420–478, 2009.

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

[16] Clemens Hofstadler. Certifying operator identities and ideal membership of noncommutative polynomials, March 2020. Masterarbeit, 2020.

[17] Jamal Hossein Poor, Clemens G. Raab, and Georg Regensburger. Algorithmic operator algebras via normal forms in tensor rings. *Journal of Symbolic Computation*, 85:247–274, 2018. 41th International Symposium on Symbolic and Alge-braic Computation (ISSAC'16).

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

[19] Philippe Malbos and Isaac Ren. Shuffle polygraphic resolutions for operads. submitted preprint, arXiv:2012.15718, December 2020.

[20] Philippe Malbos and Isaac Ren. Completion in operads via essential syzygies. In *Proceedings of the 46th International Symposium on Symbolic and Algebraic Computation*, ISSAC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[21] Martin Markl and Elisabeth Remm. (Non-)Koszulness of operads for n-ary algebras, galgalim and other curiosities. *J. Homotopy Relat. Struct.*, 10(4):939–969, 2015.

[22] H. Michael Möller, Teo Mora, and Carlo Traverso. Gröbner bases computation using syzygies. In *Papers from the International Symposium on Symbolic and Algebraic Computation*, ISSAC '92, page 320–328, New York, NY, USA, 1992. Association for Computing Machinery.

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

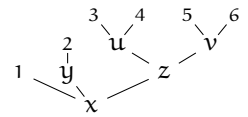
[24] Maurice Nivat. Congruences parfaites et quasi-parfaites. In *Séminaire P. Dubreil, 25e année (1971/72), Algèbre, Fasc. 1, Exp. No. 7*, page 9. Secrétariat Mathématique, Paris, 1973.

[25] Xingqiang Xiu. *Non-commutative Gröbner Bases and Applications*. PhD thesis, Universität Passau, 2012.

Appendix

As an illustration, we execute algorithm F4 on an ORS presenting the anti-associative operad. First, we introduce some notations.

Preliminaries. We represent monomials by planar trees with numbered inputs. For instance,

$w =$ 

is a monomial where the arities are $\mathrm{AR}(x) = 3$, $\mathrm{AR}(y) = 1$, and $\mathrm{AR}(z) = \mathrm{AR}(u) = \mathrm{AR}(v) = 2$. The *weight* of a monomial u is the number of its inner vertices. For instance, $|w| = 5$.

Let P be a collection. For $x \in P(k)$, $y \in P(n)$, and $1 \leqslant i \leqslant k$, denote by

$$x \circ_i y := x \circ (\varepsilon, \dots, \varepsilon, \underset{i}{y}, \varepsilon, \dots, \varepsilon)$$

the *elementary composition* of x and y.

Example. Consider the following ORS that presents the *anti-associative operad* [21]

$$X := \langle x \in X(2) \mid f : x \circ_1 x \to -x \circ_2 x \rangle.$$

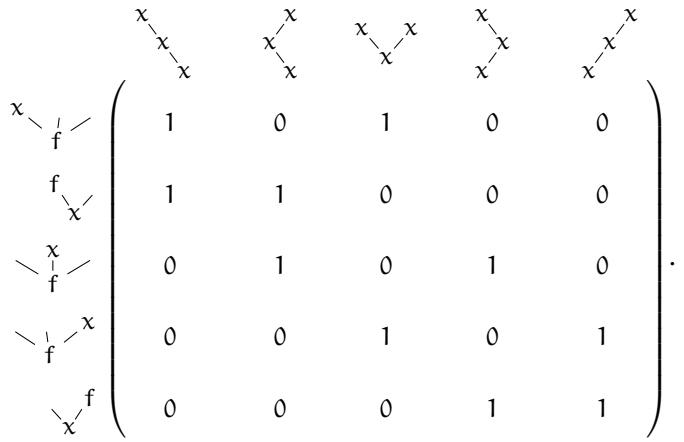
Let us study the execution of algorithm F4 with:

1. the confluence obstruction map that selects essential branchings, [20],
2. the selection strategy that selects the branchings of lowest weight,
3. the reverse path-lexicographic monomial order $\prec$, [3],
4. the reduction strategy σ given by taking the smallest rewriting monomial for the context path-lexicographic order defined in [20].

At the first iteration of the algorithm F4, there is one essential branching $(f \circ_1 x, x \circ_1 f)$. The algorithm GetRM applied to $(X, \{f \circ_1 x, x \circ_1 f\})$ returns the set

$$R' = \{x \circ_1 f, f \circ_2 x, x \circ_2 f, f \circ_1 x, f \circ_3 x\}.$$

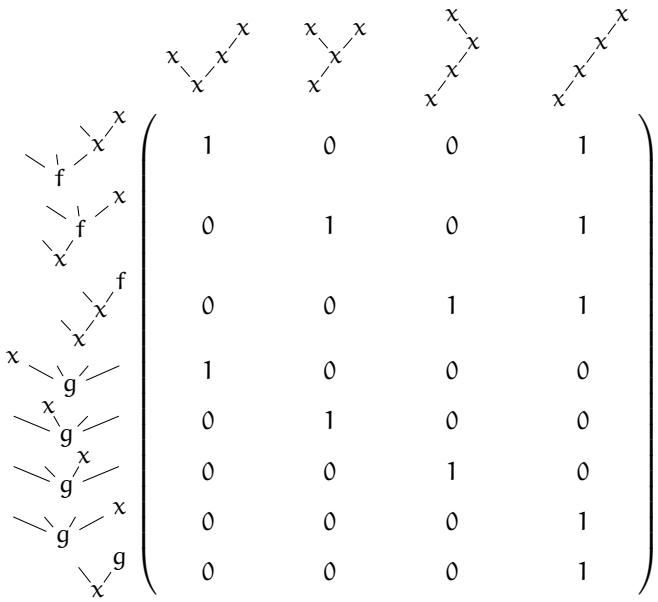
Then the matrix $M_{R'}$ is of the following form



where the columns are ordered by reverse path-lexicographic order. We check that $\mathsf{RowReduce}(M_{R'})$ is the 5×5 identity matrix, and that $\mathsf{Reduction}(X, \{f \circ_1 x, x \circ_1 f\})$ returns one rewriting rule, $g : x \circ_2 (x \circ_2 x) \to 0$, which we add to the ORS. At the next iteration, there are four essential branchings:

$$P := \{(f \circ_3 (x \circ_2 x), g \circ_1 x), (x \circ (f \circ_3 x), g \circ_2 x), (x \circ_2 (x \circ_2 f), g \circ_3 x), (x \circ_2 g, g \circ_4 x)\}.$$

Using the selection strategy, we once again select all branchings. The matrix $M_{\mathsf{GetRM}(\sigma)(X,P)}$ is



Since each column corresponds to the source of a rewriting monomial in R', the algorithm $\mathsf{Reduction}$ cannot produce new rewriting rules. Thus, the procedure $\mathsf{F4}$ terminates and the final convergent presentation is

$$\langle\, x \in X(2) \mid f : x \circ_1 x \to -x \circ_2 x,\ g : x \circ_2 (x \circ_2 x) \to 0 \,\rangle.$$

Formalized Signature Extension Results for Confluence, Commutation and Unique Normal Forms *

Alexander Lochmann[1], Fabian Mitterwallner[1], Aart Middeldorp[1]

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

Abstract

Ground-confluence and confluence do not coincide. However, for the class of left-linear right-ground TRSs confluence can be reduced to ground-confluence by extending the signature with fresh constants. We present a formalization in Isabelle/HOL of a more general result, for linear variable-separated rewrite systems. From this formalization we obtain a sound procedure to decide confluence, commutation and unique normal forms of such systems. We implemented this procedure in the decision tool FORT-h, which also can produce machine checkable proofs, and in the certifier FORTify to validate these.

1 Introduction

Dauchet and Tison [2] proved the decidability of the *first-order theory of rewriting* for the class of ground rewrite systems. The recent tool FORT-h [6] implements an extension of the decision procedure for the larger class of linear variable-separated rewrite systems. FORT-h is capable of producing certificates that witness the steps in the decision procedure. These certificates are validated by FORTify [6], a verified Haskell program obtained from the Isabelle/HOL formalization of the underlying theory reported in [5].

The decision procedure is based on tree automata techniques and hence is restricted to properties on ground terms. In this paper we are concerned with extending FORT-h and FORTify to deal with confluence-related properties on arbitrary terms. These include commutation (COM) and unique normal forms with respect to conversion (UNC) and reduction (UNR). This allows the combination of these tools to be the first participant that produces provably correct answers in the categories COM, UNC and UNR of the Confluence Competition (CoCo).

We assume familiarity with (first-order) term rewriting [1], but do not impose the usual variable restrictions on rewrite rules. In the next section we present the formalized signature extension results that allow to reduce confluence-related properties to properties on ground terms. Section 3 explains the changes made to FORT-h and FORTify. We conclude in Section 4 with suggestions for future research.

2 Theory

We start this section by recalling the results of [3, 7, 8] concerning the reduction of confluence-related properties to their ground versions. The first lemma is from [7, 8]. Here $\mathcal{P}$ consists of

*This work is supported by the Austrian Science Fund (FWF) project P30301.

the following properties

$$\begin{aligned}
\mathsf{CR}: &\quad \forall s \forall t \forall u\ (s \to^* t \wedge s \to^* u \implies t \downarrow u) && \text{confluence}\\
\mathsf{SCR}: &\quad \forall s \forall t \forall u\ (s \to^* t \wedge s \to^* u \implies \exists v\ (t \to^= v \wedge u \to^* v)) && \text{strong confluence}\\
\mathsf{WCR}: &\quad \forall s \forall t \forall u\ (s \to t \wedge s \to u \implies t \downarrow u) && \text{local confluence}\\
\mathsf{NFP}: &\quad \forall s \forall t \forall u\ (s \to^* t \wedge s \to^! u \implies t \to^! u) && \text{normal form property}\\
\mathsf{UNR}: &\quad \forall s \forall t \forall u\ (s \to^! t \wedge s \to^! u \implies t = u) && \text{unique normal forms wrt reduction}\\
\mathsf{UNC}: &\quad \forall t \forall u\ (t \leftrightarrow^* u \wedge \mathsf{NF}(t) \wedge \mathsf{NF}(u) \implies t = u) && \text{unique normal forms wrt conversion}
\end{aligned}$$

For a property $P \in \mathcal{P}$, $\mathsf{G}P$ denotes the property P restricted to ground terms.

Lemma 1. *Let $\mathcal{R}$ be a left-linear right-ground* TRS *over a signature $\mathcal{F}$ that contains at least one constant.*

1. $(\mathcal{F}, \mathcal{R}) \vDash P \iff (\mathcal{F} \uplus \{c\}, \mathcal{R}) \vDash \mathsf{G}P$ *for all* $P \in \mathcal{P} \setminus \{\mathsf{UNC}\}$
2. $(\mathcal{F}, \mathcal{R}) \vDash \mathsf{UNC} \iff (\mathcal{F} \uplus \{c, d\}, \mathcal{R}) \vDash \mathsf{GUNC}$
3. *If $\mathcal{R}$ is ground or $\mathcal{F}$ is monadic then* $(\mathcal{F}, \mathcal{R}) \vDash P \iff (\mathcal{F}, \mathcal{R}) \vDash \mathsf{G}P$ *for all* $P \in \mathcal{P}$. □

The constants c and d are assumed to be fresh (i.e., $c, d \notin \mathcal{F}$) throughout this paper. A signature is monadic if every function symbol has arity at most one. A formalization in Isabelle/HOL of the third item has been reported in [3].

Definition 2. A TRS $\mathcal{R}$ is *variable-separated* if $\mathcal{V}\mathsf{ar}(l) \cap \mathcal{V}\mathsf{ar}(r) = \varnothing$ for all $l \to r \in \mathcal{R}$.

We emphasize that the usual restriction $\mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(l)$ is not imposed on these systems. For *linear* variable-separated TRSs the first item of Lemma 1 does not hold. In [3] a (non-formalized) proof is presented that two fresh constants are sufficient to reduce confluence to ground confluence.

Lemma 3 ([3, Theorem 6.4])**.** *If $\mathcal{R}$ is a linear variable-separated* TRS *over a signature $\mathcal{F}$ then* $(\mathcal{F}, \mathcal{R}) \vDash \mathsf{CR} \iff (\mathcal{F} \uplus \{c, d\}, \mathcal{R}) \vDash \mathsf{GCR}$. □

The necessity of adding two fresh constants follows from the following example from [3].

Example 4. Consider the linear variable-separated TRS $\mathcal{R}$ consisting of the single rule $\mathsf{a} \to x$ over the signature $\mathcal{F} = \{\mathsf{a}\}$. Since $x \ {}_{\mathcal{R}}\!\leftarrow \mathsf{a} \to_{\mathcal{R}} y$ with distinct variables x and y, $\mathcal{R}$ is not confluent. Ground-confluence holds trivially as $\mathsf{a} \to_{\mathcal{R}} \mathsf{a}$ is the only rewrite step between ground terms. Adding a single fresh constant b does not destroy ground-confluence ($\mathsf{a} \to_{\mathcal{R}} \mathsf{a}$ and $\mathsf{a} \to_{\mathcal{R}} \mathsf{b}$ are the only steps). By adding a second fresh constant c, ground-confluence is lost: $\mathsf{b} \ {}_{\mathcal{R}}\!\leftarrow \mathsf{a} \to_{\mathcal{R}} \mathsf{c}$.

We generalize Lemma 3 to commutation (COM) and unique normal forms (UNC and UNR), where

$$\mathsf{COM}: \quad \forall s \forall t \forall u\ (s \to^*_{\mathcal{R}} t \wedge s \to^*_{\mathcal{S}} u \implies \exists v\ (t \to^*_{\mathcal{S}} v \wedge u \to^*_{\mathcal{R}} v)) \qquad \text{commutation}$$

The proof below is formalized. In the proof we restrict attention to rewrite sequences that involve a root step. Root steps are important since they permit the use of two arbitrary substitutions on the left and right of rule used in the root step, due to variable separation of the rules. Therefore we start with a preliminary result (Lemma 6) which provides abstract conditions that permit this restriction. We write $\to^{*\epsilon*}_{\mathcal{R}}$ for the relation $\to^*_{\mathcal{R}} \cdot \to^{\epsilon}_{\mathcal{R}} \cdot \to^*_{\mathcal{R}}$. The proof of Lemma 6 is obtained by a straightforward induction on the term structure and the multi-hole context closure of the rewrite relation, and is omitted.

Definition 5. A binary predicate P on terms over a given signature $\mathcal{F}$ is closed under *multi-hole contexts* if $P(C[s_1,\dots,s_n],C[t_1,\dots,t_n])$ holds whenever C is a multi-hole context over $\mathcal{F}$ with $n \geqslant 0$ holes and $P(s_i,t_i)$ holds for all $1 \leqslant i \leqslant n$.

Lemma 6. *Let $\mathcal{A}$ and $\mathcal{B}$ be* TRS*s over the same signature $\mathcal{F}$ and let P be a binary predicate that is closed under multi-hole contexts over $\mathcal{F}$.*

1. *If $P(s,t)$ for all terms s and t such that $s \to_{\mathcal{A}}^{*\epsilon*} t$ then $P(s,t)$ for all terms s and t such that $s \to_{\mathcal{A}}^{*} t$.*
2. *If $P(s,t)$ for all terms s and t such that $s \to_{\mathcal{A}}^{*\epsilon*} \cdot \to_{\mathcal{B}}^{*} t$ or $s \to_{\mathcal{A}}^{*} \cdot \to_{\mathcal{B}}^{*\epsilon*} t$ then $P(s,t)$ for all terms s and t such that $s \to_{\mathcal{A}}^{*} \cdot \to_{\mathcal{B}}^{*} t$.* □

We show how Lemma 6 is instantiated for the properties of interest.

- For UNC we use part 1 with $P_1(s,t)\colon \mathsf{NF}(s) \wedge \mathsf{NF}(t) \implies s = t$ and $\mathcal{R} \cup \mathcal{R}^{-}$ for $\mathcal{A}$.
- For UNR we use part 2 with the same predicate P_1 and $\mathcal{R}^{-}$ for $\mathcal{A}$ and $\mathcal{R}$ for $\mathcal{B}$.
- For COM we use part 2 with $P_2(s,t)\colon s \to_{\mathcal{S}}^{*} \cdot \to_{\mathcal{R}^{-}}^{*} t$ and $\mathcal{R}^{-}$ for $\mathcal{A}$ and $\mathcal{S}$ for $\mathcal{B}$.

Lemma 7. *The properties P_1 and P_2 are closed under multi-hole contexts.* □

The next lemma is a key result. It allows the removal of introduced fresh constants while preserving the reachability relation. Note that variable-separation is not required.

Lemma 8. *Let $\mathcal{R}$ be a linear TRS over a signature $\mathcal{F}$ that contains a constant c which does not appear in $\mathcal{R}$. If $s \to_{\mathcal{R}}^{*} t$ with $c \in \mathcal{F}\mathsf{un}(s) \setminus \mathcal{F}\mathsf{un}(t)$ then $s[u]_p \to_{\mathcal{R}}^{*} t$ using the same rewrite rules at the same positions, for all terms u and positions $p \in \mathcal{P}\mathsf{os}(s)$ such that $s|_p = c$.*

The restriction to linear TRSs can also be lifted, at the expense of a more complicated replacement function and proof. Since the decision procedure implemented in FORT-h relies on linearity and variable-separation, we present a simple proof for linear TRSs. Due to calculations involving positions, the formalization in Isabelle/HOL was anything but simple.

Proof. We use induction on the length of $s \to_{\mathcal{R}}^{*} t$. If this length is zero then there is nothing to show as $\mathcal{F}\mathsf{un}(s) \setminus \mathcal{F}\mathsf{un}(t) = \varnothing$. Suppose $s \to_{\mathcal{R}} v \to_{\mathcal{R}}^{*} t$ and write $s = C[\ell\sigma] \to_{\mathcal{R}} C[r\sigma] = v$. Let p' be the position of the hole in C and let $p \in \mathcal{P}\mathsf{os}(s)$ with $s|_p = c$. We distinguish two cases.

If $p' \parallel p$ then $s[u]_p = (C[u]_p)[\ell\sigma]_{p'} \to_{\mathcal{R}} v'$ with $v' = (C[u]_p)[r\sigma]_{p'}$. Since $v|_p = C|_p = c$ we can apply the induction hypothesis to $v \to_{\mathcal{R}}^{*} t$. This yields $v' \to_{\mathcal{R}}^{*} t$ and hence $s[u]_p \to_{\mathcal{R}}^{*} t$ as desired.

In the remaining case, $p' \leqslant p$. From $s|_p = c$ and the fact that c does not appear in $\mathcal{R}$ we infer that there exists a variable $y \in \mathcal{V}\mathsf{ar}(\ell)$ such that $c \in \mathcal{F}\mathsf{un}(\sigma(y))$. Let q be the (unique) position of y in ℓ and consider the substitution

$$\tau(x) = \begin{cases} \sigma(y)[u]_{q'} & \text{if } x = y \\ \sigma(x) & \text{otherwise} \end{cases}$$

Here $q' = p \backslash (p'q)$ is the position of c in $\sigma(y)$. If $y \notin \mathcal{V}\mathsf{ar}(r)$ then $v = C[r\sigma] = C[r\tau]$ and thus $s[u]_p = C[\ell\tau] \to_{\mathcal{R}} C[r\tau] = v \to_{\mathcal{R}}^{*} t$. If $y \in \mathcal{V}\mathsf{ar}(r)$ then there exists a unique position $q'' \in \mathcal{P}\mathsf{os}(r)$ such that $r|_{q''} = y$. So $v|_{p'q''q'} = c$ and we obtain $s[u]_p = C[\ell\tau] \to_{\mathcal{R}} C[r\tau] = v[u]_{p'q''q'} \to_{\mathcal{R}}^{*} t$ from the induction hypothesis. □

Using the preceding two lemmata, the main result easily follows.

Lemma 9. *Linear variable-seperated* TRS*s* $\mathcal{R}$ *and* $\mathcal{S}$ *over a common signature* $\mathcal{F}$ *commute if and only if* $\mathcal{R}$ *and* $\mathcal{S}$ *ground-commute over* $\mathcal{F} \uplus \{c, d\}$.

Proof. First we prove the if direction. So suppose $\mathcal{R}$ and $\mathcal{S}$ ground-commute on terms in $\mathcal{T}(\mathcal{F} \uplus \{c, d\})$. In order to conclude that $\mathcal{R}$ and $\mathcal{S}$ commute on terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$, according to Lemma 6 it suffices to show the inclusions

$$\to_{\mathcal{R}^-}^{*\epsilon*} \cdot \to_{\mathcal{S}}^{*} \subseteq \to_{\mathcal{S}}^{*} \cdot \to_{\mathcal{R}^-}^{*} \qquad \to_{\mathcal{R}^-}^{*} \cdot \to_{\mathcal{S}}^{*\epsilon*} \subseteq \to_{\mathcal{S}}^{*} \cdot \to_{\mathcal{R}^-}^{*}$$

on terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Suppose $s \to_{\mathcal{R}^-}^{*\epsilon*} \cdot \to_{\mathcal{S}}^{*} t$. Let the substitution σ_c map all variables to c and let σ_d map all variables to d. Since rewriting is closed under substitutions and the variable-separated rule used in the root step $\to_{\mathcal{R}^-}^{\epsilon}$ allows changing the substitution, we obtain $s\sigma_c \to_{\mathcal{R}^-}^{*\epsilon*} \cdot \to_{\mathcal{S}}^{*} t\sigma_d$. From ground commutation we obtain $s\sigma_c \to_{\mathcal{S}}^{*} \cdot \to_{\mathcal{R}^-}^{*} t\sigma_d$. Note that s and t are terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and hence do not contain the constants c and d. Therefore, $d \notin \mathcal{F}\mathsf{un}(s\sigma_c)$ and $c \notin \mathcal{F}\mathsf{un}(t\sigma_d)$. As a consequence, repeated applications of Lemma 8 transform $s\sigma_c \to_{\mathcal{S}}^{*} \cdot \to_{\mathcal{R}^-}^{*} t\sigma_d$ into a sequence $s \to_{\mathcal{S}}^{*} \cdot \to_{\mathcal{R}^-}^{*} t$ in which c and d do not appear, proving the first inclusion. Note that in our setting TRSs are closed under rule reversal. Hence we can apply Lemma 8 in both directions, which allows us to remove the constant d from the term t. The second inclusion $\to_{\mathcal{R}^-}^{*} \cdot \to_{\mathcal{S}}^{*\epsilon*} \subseteq \to_{\mathcal{S}}^{*} \cdot \to_{\mathcal{R}^-}^{*}$ is obtained in the same way.

For the only-if direction we assume that $\mathcal{R}$ and $\mathcal{S}$ commute on terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and use Lemma 6 to establish the commutation of $\mathcal{R}$ and $\mathcal{S}$ on terms in $\mathcal{T}(\mathcal{F} \uplus \{c, d\})$. We prove the first inclusion. The second inclusion follows then by a symmetric argument. So let $s \to_{\mathcal{R}^-}^{*\epsilon*} \cdot \to_{\mathcal{S}}^{*} t$ and consider the following mapping $\phi\colon \mathcal{T}(\mathcal{F} \uplus \{c, d\}) \to \mathcal{T}(\mathcal{F}, \{x, y\})$:

$$\phi(t) = \begin{cases} x & \text{if } t = c \\ y & \text{if } t = d \\ f(\phi(t_1), \ldots, \phi(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Here x and y are distinct variables in $\mathcal{V}$. A straightforward induction proof shows $\phi(u) \to_{\mathcal{R}^-}^{*} \phi(v)$ whenever $u \to_{\mathcal{R}^-}^{*} v$, for all $u, v \in \mathcal{T}(\mathcal{F} \uplus \{c, d\})$. The same holds for $\mathcal{S}$. Hence, the given sequence from s to t is transformed into $\phi(s) \to_{\mathcal{R}^-}^{*\epsilon*} \cdot \to_{\mathcal{S}}^{*} \phi(t)$. Since c and d do not appear in the transformed sequence, we obtain $\phi(s) \to_{\mathcal{S}}^{*} \cdot \to_{\mathcal{R}^-}^{*} \phi(t)$ from the commutation of $\mathcal{R}$ and $\mathcal{S}$. Define the substitution $\tau = \{x \mapsto c, y \mapsto d\}$. Since rewriting is closed under substitution, $s = \phi(s)\tau \to_{\mathcal{S}}^{*} \cdot \to_{\mathcal{R}^-}^{*} \phi(t)\tau = t$. □

The proofs for the unique normal form properties (UNC and UNR) are obtained in a similar manner.

Lemma 10. *Let* $\mathcal{R}$ *be a left-linear variable-separated* TRS *over a signature* $\mathcal{F}$ *that contains at least one constant.*

- $(\mathcal{F}, \mathcal{R}) \vDash \mathsf{UNR} \iff (\mathcal{F} \uplus \{c, d\}, \mathcal{R}) \vDash \mathsf{GUNR}$.
- $(\mathcal{F}, \mathcal{R}) \vDash \mathsf{UNC} \iff (\mathcal{F} \uplus \{c, d\}, \mathcal{R}) \vDash \mathsf{GUNC}$. □

3 FORT-h and FORTify

The overall design of FORT-h and FORTify is shown in Figure 1. If FORT-h does not time out, it produces a certificate in the certificate language that is formally described in [6, Section 4].

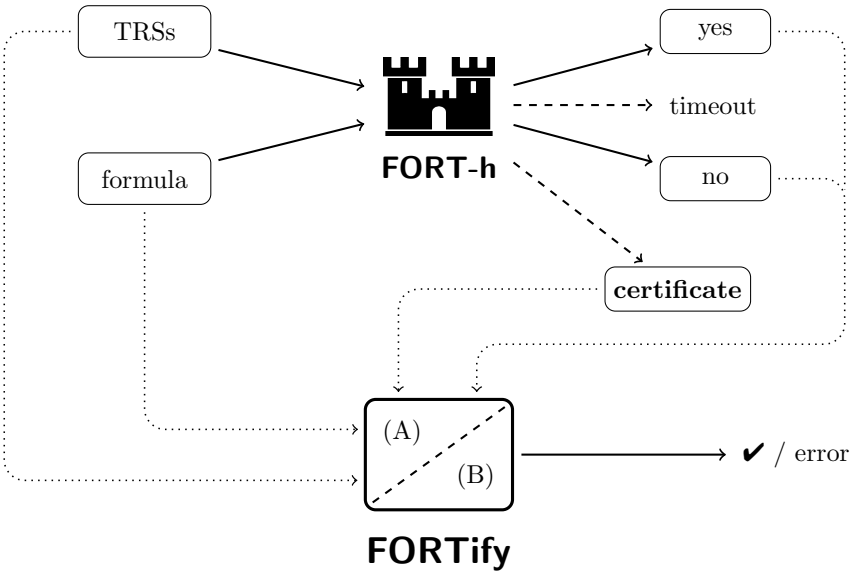


Figure 1: FORT-h and FORTify.

Certificates can be viewed as a recipe for the certifier to perform certain operations on tree automata and formulas in order to confirm the yes/no claim of FORT-h. The certifier is the verified Haskell code base that is generated by Isabelle's code generation facility, corresponding to module (B) of FORTify. Module (A) contains a Haskell parser to translate strings representing formulas (TRSs, signatures, certificates) to semantically equivalent objects in the data types obtained from the generated code in module (B). The reader is referred to [6] for further details.

Here we briefly describe the required changes to this setup in order to accommodate the results mentioned in the preceding sections.

FORT-h already had support for some properties on open terms [6] based only on Lemma 3. If the input formula was one of the predefined macros for a property on open terms (e.g. CR), it would execute the decision procedure with the signature extended by two constants on the formula of the corresponding ground property (e.g. GCR). To improve the performance of the decision procedure we implemented the optimizations described in Lemma 1. This means the number of additional constants now depends on the properties of the input TRS, which in some cases leads to smaller signatures, therefore leading to faster decisions by the tool.

The more interesting changes relate to FORTify. Since the certificate serves as a proof that a formula holds for ground terms, we chose to keep the certificate format unchanged. The signature extension described in Lemmata 1, 3 and 9 were implemented as a preprocessing step of the formula which, just like FORT-h, checks if the input formula is a property on open terms. If that is the case, the signature is extended and the formula set to the corresponding ground property. Here care has to be taken that both FORT-h and FORTify use the same definitions for their ground property, since this formula has to match the one in the certificate. The choice to keep the certificate unchanged also means that the interface between FORT-h and FORTify remains unchanged and FORTify is fully backwards compatible. Note that this preprocessing step is implemented in module (A) of FORTify (see Figure 1) by hand, hence is

not code generated from the formalization.

4 Conclusion

We showed that commutation of linear variable-separated TRSs reduces to ground-commutation after the signature is extended with two fresh constants. (This is not to be confused with signature extension results for commutation, which are studied in [4,9].) The proof is formalized in Isabelle/HOL and can be obtained from the website

https://fortissimo.uibk.ac.at/iwc2021

accompanying this paper. Precompiled binaries of the new versions of FORT-h and FORTify are available from the same site. A similar formalized proof for NFP is expected soon.

The current implementation of FORTify supports certifying decisions of the properties UNR, UNC, CR, and COM of FORT-h. At the moment these properties must appear at the root of the input formula. This restriction comes from the underlying decision procedure presented in [5] in which the signature is assumed to be fixed. Possible future work is to permit these properties to appear within a formula. This would allow certifying results for a formula like $\mathsf{GCR} \wedge \neg\mathsf{CR}$. FORT-h already has support for this, but the results cannot be certified.

Another improvement would be moving the signature extension procedure from module (A) into the formally verified module (B). While this would necessarily change the interface between (A) and (B), the certificate format could still remain unchanged for backwards compatibility.

References

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

[2] Max Dauchet and Sophie Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th LICS*, pages 242–248, 1990. doi:10.1109/LICS.1990.113750.

[3] Bertram Felgenhauer, Aart Middeldorp, T. V. H. Prathamesh, and Franziska Rapp. A verified ground confluence tool for linear variable-separated rewrite systems in Isabelle/HOL. In *Proc. 8th CPP*, pages 132–143, 2019. doi:10.1145/3293880.3294098.

[4] Nao Hirokawa. Commutation and signature extensions. In *Proc. 4th IWC*, pages 23–27, 2015.

[5] Alexander Lochmann, Aart Middeldorp, Fabian Mitterwallner, and Bertram Felgenhauer. A verified decision procedure for the first-order theory of rewriting for linear variable-separated rewrite systems. In *Proc. 10th CPP*, pages 250–263, 2021. doi:10.1145/3437992.3439918.

[6] Fabian Mitterwallner, Alexander Lochmann, Aart Middeldorp, and Bertram Felgenhauer. Certifying proofs in the first-order theory of rewriting. In *Proc. 27th TACAS*, volume 12652 of *LNCS*, pages 127–144, 2021. doi:10.1007/978-3-030-72013-1_7.

[7] Franziska Rapp and Aart Middeldorp. Confluence properties on open terms in the first-order theory of rewriting. In *Proc. 5th IWC*, pages 26–30, 2016.

[8] Franziska Rapp and Aart Middeldorp. FORT 2.0. In *Proc. 9th IJCAR*, volume 10900 of *LNAI*, pages 81–88, 2018. doi:10.1007/978-3-319-94205-6_6.

[9] Kiraku Shintani and Nao Hirokawa. CoLL: A confluence tool for left-linear term rewrite systems. In *Proc. 25th CADE*, volume 9195 of *LNCS*, pages 127–136, 2015. doi:10.1007/978-3-319-21401-6_8.

Evaluation in the computational calculus is non-confluent

Claudia Faggian[1], Giulio Guerrieri[2], and Riccardo Treglia[3]

[1] Université de Paris, IRIF, CNRS, F-75013 Paris, France
faggian@irif.fr
[2] University of Bath, Department of Computer Science, Bath, UK
giulio.guerrieri@gmail.com
[3] Università di Torino, Department of Computer Science, Turin, Italy
riccardo.treglia@unito.it

Abstract

In Moggi's computational calculus, reduction is the contextual closure of the rules obtained by orienting three monadic laws. In the literature, evaluation is usually defined as the closure under weak contexts (no reduction under binders): $\mathsf{E} = \langle\rangle \mid \mathsf{let}\, x := \mathsf{E}\, \mathsf{in}\, M$.

We show that, when considering all the monadic rules, weak reduction is non-deterministic, non-confluent, and normal forms are not unique. However, when interested in returning a value (convergence), the only necessary monadic rule is β, whose evaluation is deterministic.

The *computational* λ*-calculus*, noted λ_c, was introduced by Moggi [11, 12, 13] as a meta-language to describe computational effects in programming languages. Since then, computational λ-calculi have been developed as foundations of programming languages, formalizing both functional and effectful features [20, 1, 16, 9, 2], in a still active line of research.

To model effectful features at a semantic level, Moggi used the categorical notion of *monad*. A monad can be equivalently presented as a Kleisli triple satisfying three identities [13, 10]. At an operational level, Moggi [11] internalized these identities into the syntax of λ_c, giving rise to three conversion rules—called *monadic laws*—that are added to the usual β and η rules.

Nowadays the literature is rich of computational calculi that refine Moggi's λ_c. Such calculi are presented in at least three different fashions: fully equational systems [9, 15] (all conversion rules are unoriented identities); hybrid systems where β (and η, if considered) are oriented rules while the monadic laws are identities on terms [2]; reduction systems where every rule is oriented [17]. Here we follow the latter approach, which brings to the fore operational aspects of reduction and evaluation which seem to have been neglected in the literature.

Indeed, in the literature of calculi with effects [9, 2], *evaluation* is usually *weak*, that is, it is not allowed in the scope of the binders (λ or let). This is the way evaluation is implemented by functional programming languages such as Haskell and OCaml. Moreover, only β and $\mathsf{let}.\beta$ are considered. However, in Moggi's λ_c and in Sabry and Wadler's [17], the reduction is full, that is, reduction is the compatible closure of all the monadic rules. When considering *all the rules*, we observe—quite unexpectedly—that evaluation (*i.e.* weak reduction) is *non-deterministic, non-confluent, and normal forms are not unique.*

Reduction and Evaluation. Here we focus on a computational λ-calculus which is standard in the literature, namely Sabry and Wadler's λ_{ml^*} [17]. This is a neat and compact refinement of Moggi's untyped λ_c [11]—the relation between the two calculi is formalized by a reflection[17].

λ_{ml^*}—which we display in Figure 1— has a two sorted syntax that separates *values* (*i.e.* variables and abstractions) and *computations*. The latter are either *let*-expressions (aka explicit substitutions, capturing monadic binding), or applications (of values to values), or coercions $[V]$ of values V into computations (corresponding to the `return` operator in Haskell).

$$\begin{array}{rrcl} \textit{Values:} & V, W & ::= & x \mid \lambda x.M \\ \textit{Computations:} & M, N & ::= & [V] \mid \mathsf{let}\, x := M \,\mathsf{in}\, N \mid VW \end{array}$$

Reduction rules:

$$\begin{array}{rrcll} (\beta) & (\lambda x.M)V & \mapsto_{\beta} & M[V/x] & \\ (\eta) & \lambda x.Vx & \mapsto_{\eta} & V & x \notin \mathsf{fv}(V) \\ (\mathsf{let}.\beta) & \mathsf{let}\, x := [V] \,\mathsf{in}\, N & \mapsto_{\mathsf{let}.\beta} & N[V/x] & \\ (\mathsf{let}.\eta) & \mathsf{let}\, x := M \,\mathsf{in}\, [x] & \mapsto_{\mathsf{let}.\eta} & M & \\ (\mathsf{let}.ass) & \mathsf{let}\, y := (\mathsf{let}\, x := L \,\mathsf{in}\, M) \,\mathsf{in}\, N & \mapsto_{\mathsf{let}.ass} & \mathsf{let}\, x := L \,\mathsf{in}\, (\mathsf{let}\, y := M \,\mathsf{in}\, N) & x \notin \mathsf{fv}(N) \end{array}$$

Figure 1: λ_{ml^*}: Syntax and Reduction

- The *reduction rules* in λ_{ml^*} are the usual β (and η) rules from Plotkin's *call-by-value* λ-calculus [14], plus the oriented version of the three *monadic laws*: $\mathsf{let}.\beta$, $\mathsf{let}.\eta$, $\mathsf{let}.ass$ (see Figure 1).
- *Reduction* $\to$ is the *contextual closure* of the reduction rules.

Following standard practice, we define *evaluation* $\underset{\mathsf{w}}{\to}_{ml^*}$ (aka *sequencing*) as the closure of the rules under *evaluation context* E:

$$\mathsf{E} \;::=\; \langle\,\rangle \mid \mathsf{let}\, x := \mathsf{E} \,\mathsf{in}\, N \qquad \text{evaluation context}$$

Informally, the operational understanding of weak reduction is that evaluating $\mathsf{let}\, x := M \,\mathsf{in}\, N$ amounts to first evaluate M until it returns a value, that is, until a computation of the form $[V]$ is reached. Then V is passed to N by substituting V for x in N, thanks to the rule $\mathsf{let}.\beta$.

Despite the prominent role that weak reduction has in the literature of calculi with effects, its reduction properties are somehow surprising. While full reduction $\to_{ml^*}$ is confluent, the closure of the rules under *evaluation context* turns out to be *non-deterministic*, *non-confluent*, and its *normal forms* are *not unique*.

Note that such issues only come from the monadic rules $\mathsf{let}.\eta$ and $\mathsf{let}.ass$ (sometimes called *identity* and *associativity*, respectively, in the literature), not from β or $\mathsf{let}.\beta$. It is worth to clarify that while the literature on computational λ-calculi often adopts weak reduction (see for instance, [9, 2], where a big-step variant is used), the rules $\mathsf{let}.ass$ and $\mathsf{let}.\eta$ are usually dealt with as *unoriented* identities—the only oriented rules being β and $\mathsf{let}.\beta$.

(Non-)Confluence. In λ_{ml^*}, the reduction $\to_{ml^*}$ is confluent, but weak reduction $\underset{\mathsf{w}}{\to}_{ml^*}$ is not. We now give some examples. For every $\gamma \in \{\beta, \eta, \mathsf{let}.\beta, \mathsf{let}.\eta, \mathsf{let}.ass\}$, the *weak* γ*-reduction* $\underset{\mathsf{w}}{\to}_{\gamma}$ is the closure of the rule $\mapsto_{\gamma}$ under *weak contexts* E.

Example 1 (Non-confluence)**.** Let M be a computation in normal form, for instance $M = xx$.

$$\begin{array}{ccc} \mathsf{let}\, y := (\mathsf{let}\, x := zz \,\mathsf{in}\, M) \,\mathsf{in}\, [y] & \xrightarrow[\mathsf{w}]{\mathsf{let}.\eta} & \mathsf{let}\, x := zz \,\mathsf{in}\, M \\ \Big\downarrow{\scriptstyle \mathsf{let}.ass,\,\mathsf{w}} & & \\ \mathsf{let}\, x := zz \,\mathsf{in}\, (\mathsf{let}\, y := M \,\mathsf{in}\, [y]) & & \end{array}$$

Both $\mathsf{let}\, x := zz \,\mathsf{in}\, M$ and $\mathsf{let}\, x := zz \,\mathsf{in}\, (\mathsf{let}\, y := M \,\mathsf{in}\, [y])$ are normal for $\underset{\mathsf{w}}{\to}_{ml^*}$ (in the latter, the $\mathsf{let}.\eta$-redex $\mathsf{let}\, y := M \,\mathsf{in}\, [y]$ cannot be fired by weak reduction), but they are distinct.

Example 2 (Non-confluence)**.** Let $R = P = Q = L = zz$ and:

$$M \;:=\; \overline{\texttt{let } z = \underline{(\texttt{let } x = (\texttt{let } y = L \texttt{ in } Q) \texttt{ in } P)} \texttt{ in } R}$$

There are two weak $\mathsf{let}.ass$-redexes, the overlined one and the underlined one. So,

$$\begin{aligned} M &\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass} \mathsf{let}\, x := (\mathsf{let}\, y := L \,\mathsf{in}\, Q) \,\mathsf{in}\, (\mathsf{let}\, z := P \,\mathsf{in}\, R) \\ &\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass} \mathsf{let}\, y := L \,\mathsf{in}\, (\mathsf{let}\, x := Q \,\mathsf{in}\, (\mathsf{let}\, z := P \,\mathsf{in}\, R)) =: M' \\ M &\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass} \mathsf{let}\, z := (\mathsf{let}\, y := L \,\mathsf{in}\, (\mathsf{let}\, x := Q \,\mathsf{in}\, P)) \,\mathsf{in}\, R \\ &\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass} \mathsf{let}\, y := L \,\mathsf{in}\, (\mathsf{let}\, z := (\mathsf{let}\, x := Q \,\mathsf{in}\, P) \,\mathsf{in}\, R) =: M'' \end{aligned}$$

Both M' are M'' are normal for $\underset{\mathsf{w}}{\to}_{ml^*}$ (in M'', the $\mathsf{let}.ass$-redex $\mathsf{let}\, z := (\mathsf{let}\, x := Q \,\mathsf{in}\, P) \,\mathsf{in}\, R$ is under the scope of a let and so cannot be fired by weak reduction), but they are distinct.

Example 3.

Non-determinism—but confluence—of $\to_{\mathsf{let}.\eta}$. Let $M = yy$ and $N = zz$:

$$\begin{array}{ccc} \mathsf{let}\, x := (\mathsf{let}\, y := (\mathsf{let}\, z := N \,\mathsf{in}\, [z]) \,\mathsf{in}\, M) \,\mathsf{in}\, [x] & \xrightarrow[\mathsf{w}]{\mathsf{let}.\eta} & \mathsf{let}\, y := (\mathsf{let}\, z := N \,\mathsf{in}\, [z]) \,\mathsf{in}\, M \\ \Big\downarrow{\scriptstyle \mathsf{let}.\eta}\ \mathsf{w} & & \Big\downarrow{\scriptstyle \mathsf{let}.\eta}\ \mathsf{w} \\ \mathsf{let}\, x := (\mathsf{let}\, y := N \,\mathsf{in}\, M) \,\mathsf{in}\, [x] & \xrightarrow[\mathsf{w}]{\mathsf{let}.\eta} & \mathsf{let}\, y := N \,\mathsf{in}\, M \end{array}$$

Summing up the situation:

1. $\underset{\mathsf{w}}{\to}_{\beta}$ and $\underset{\mathsf{w}}{\to}_{\mathsf{let}.\beta}$ and $\underset{\mathsf{w}}{\to}_{\beta,\,\mathsf{let}.\beta} := \underset{\mathsf{w}}{\to}_{\beta} \cup \underset{\mathsf{w}}{\to}_{\mathsf{let}.\beta}$ are deterministic.
2. $\underset{\mathsf{w}}{\to}_{\mathsf{let}.\eta}$ is non-deterministic, but it is confluent.
3. $\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass}$ is non-deterministic, non-confluent and normal forms are not unique.
4. $\underset{\mathsf{w}}{\to}_{\mathsf{let}.ass} \cup \underset{\mathsf{w}}{\to}_{\mathsf{let}.\beta} \cup \underset{\mathsf{w}}{\to}_{\beta}$ is non-deterministic, non-confluent and normal forms are not unique.
5. $\underset{\mathsf{w}}{\to}_{ml^*}$ is non-deterministic, non-confluent and normal forms are not unique.

(Non-)Factorization. Another remarkable aspect making the reduction theory for λ_{ml^*} (and for other computational λ-calculi) tricky to study is the lack of *factorization*, which is the simplest possible form of *standardization*.

In Plotkin's call-by-value λ-calculus [14] (which can be seen as the restriction of λ_{ml^*} where the reduction is generated only by the β-rule), weak reduction satisfies factorization, that is any reduction sequence can be *reorganized* as weak steps followed by non-weak steps:

$$\to_{\beta}^{*} \subseteq \underset{\mathsf{w}}{\to}_{\beta}^{*} \cdot \underset{\neg\mathsf{w}}{\to}_{\beta}^{*} \tag{1}$$

But in λ_{ml^*} (and similar computational λ-calculi), weak factorization *does not hold*. The problem is here the $\mathsf{let}.\eta$ rule, as shown by the following counterexample, due to van Oostrom [19].

Example 4 (Non-factorization [19])**.** Consider

$$M := \mathsf{let}\, y := (zz)\, \mathsf{in}\, (\mathsf{let}\, x := [y]\, \mathsf{in}\, [x]) \;\; \underset{\neg \mathsf{w}}{\rightarrow}_{\mathsf{let}.\eta} \;\; \mathsf{let}\, y := (zz)\, \mathsf{in}\, [y] \;\; \underset{\mathsf{w}}{\rightarrow}_{\mathsf{let}.\eta} \;\; (zz) =: N$$

Weak steps are not possible from M, so it is *impossible* to factorize the reduction form M to N as $M \underset{\mathsf{w}}{\rightarrow}^*_{ml^*} \cdot \underset{\neg \mathsf{w}}{\rightarrow}^*_{ml^*} N$.

A bridge between Evaluation and Reduction. On the one hand, computational λ-calculi such as λ_{ml^*} have an unrestricted *non-deterministic reduction* that generates the equational theory of the calculus, studied for foundational and semantic purposes. On the other hand, *weak reduction* has a prominent role in the literature of computational λ-calculi, because it models an ideal programming language. Indeed, when restricted to closed terms (which are the terms corresponding to programs), normal forms of weak reduction coincide with values; and when restricted to β and $\mathsf{let}.\beta$ steps, weak reduction is deterministic and corresponds to an abstract machine, implementing a programming language. It is then natural to wonder what is the relation between reduction and evaluation.

In Plotkin's call-by-value λ-calculus [14], the following *convergence result* provides a bridge between reduction and evaluation: if a term M β-reduces to a value, then M only needs *weak* β-reduction to reach a value.

$$M \to^*_\beta V \text{ (for some value } V) \iff M \underset{\mathsf{w}}{\rightarrow}^*_\beta V' \text{ (for some value } V') \tag{2}$$

In λ_{ml^*}, despite several drawbacks of weak reduction, we can still prove a *convergence* result similar to (2) relating reduction and evaluation: to reach a value in λ_{ml^*}, weak β-steps and weak $\mathsf{let}.\beta$-steps suffice.

Theorem 5 (Convergence)**.** *Let M be a computation in λ_{ml^*} and let $\to_{ml^*_-} := \to_{ml^*} \setminus \to_\eta$.*

$$M \to^*_{ml^*_-} [V] \text{ (for some value } V) \iff M \underset{\mathsf{w}}{\rightarrow}^*_{\beta,\, \mathsf{let}.\beta} [V'] \text{ (for some value } V') \tag{3}$$

Because of the issues which we have presented, this result is non-trivial. We obtain it via the study of a calculus recently introduced by de'Liguoro and Treglia's, namely the *computational core* $\lambda_{\copyright}$ [4]. $\lambda_{\copyright}$ has the same issues, but a different syntax, which is more closely related to calculi inspired by linear logic [18, 5, 8, 6], whose properties and tools we can then use. The analysis of the reduction theory of $\lambda_{\copyright}$ is carried-out in [7]. We then transfer the convergence of $\lambda_{\copyright}$ to that of λ_{ml^*}, via a rather sophisticated analysis of the translation.

Conclusion. Convergence in λ_{ml^*} relates full reduction to evaluation, and provides a theoretical justification to the following facts:

1. functional programming languages with computational effects use weak reduction as evaluation mechanism; indeed, weak reduction is enough to *return values*.

2. in computational λ-calculi, when interested in returning a value, the only *rules* of interest *for weak reduction* are β and $\mathsf{let}.\beta$—which are deterministic and do not have unpleasant rewriting properties—while the rules $\mathsf{let}.ass$ and $\mathsf{let}.\eta$ can be safely considered as unoriented identities external to the reduction.

IWC2021 vs IWC2020. We present work which has been developed after de'Liguoro and Treglia's presentation at IWC20 [3], and thanks to the interactions there. The developments benefited of the discussion at the workshop, in particular of subsequent crucial comments by Vincent van Oostrom [19], and of new collaborations prompted there. In [3], preliminary—and incomplete—work on weak factorization for de'Liguoro and Treglia's computational calculus $\lambda_{\copyright}$ [4] was presented. Such a work has then evolved in the analysis of the reduction theory for $\lambda_{\copyright}$ in [7]. One may wonder if the properties discovered there are specific to that specific calculus, or how that relates to the literature of computational calculi.

Here, we focus on *mainstream* and well-established formalizations of the computational calculus. We consider a standard calculus which is well-studied in the literature, namely Sabry and Wadler's λ_{ml^*} [17]. We show that the properties of non-confluence and non-factorization of evaluation which are studied in [7] actually do hold also in λ_{ml^*}—and in fact in any calculus in which the monadic rules are oriented. We find this fact quite surprising, and worth to be explicitly stated. To our knowledge, it does not appear in the literature.

Furthermore, we are able to show that the convergence result which is established in [7] transfers to λ_{ml^*}, even though the translation between the two calculi does not directly preserve weak reduction (a more sophisticated analysis is needed).

References

[1] Nick Benton, John Hughes, and Eugenio Moggi. Monads and effects. In *Applied Semantics, International Summer School, APPSEM 2000*, volume 2395 of *Lecture Notes in Computer Science*, pages 42–122. Springer, 2002.

[2] Ugo Dal Lago, Francesco Gavazzo, and Paul B. Levy. Effectful Applicative Bisimilarity: Monads, Relators, and Howe's Method. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017.

[3] Ugo de'Liguoro and Riccardo Treglia. On the reduction of the type-free computational lambda-calculus. Presentation at the 9th International Workshop on Confluence, 2020.

[4] Ugo de'Liguoro and Riccardo Treglia. The untyped computational λ-calculus and its intersection type discipline. *Theor. Comput. Sci.*, 846:141–159, 2020.

[5] Thomas Ehrhard and Giulio Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming (PPDP 2016)*, pages 174–187. ACM, 2016.

[6] Claudia Faggian and Giulio Guerrieri. Factorization in call-by-name and call-by-value calculi via linear logic. In *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021*, volume 12650 of *Lecture Notes in Computer Science*, pages 205–225. Springer, 2021.

[7] Claudia Faggian, Giulio Guerrieri, Ugo de'Liguoro, and Riccardo Treglia. On reduction and normalization in the computational core. *CoRR*, abs/2104.10267, 2021. Submitted to Math. Struct. Comp. Sci., special issue of IWC 2020.

[8] Giulio Guerrieri and Giulio Manzonetto. The bang calculus and the two Girard's translations. In *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications (Linearity-TLLA 2018)*, volume 292 of *EPTCS*, pages 15–30, 2019.

[9] Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Information and Computation*, 185(2):182 – 210, 2003.

[10] Saunders MacLane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer, 2 edition, 1997.

[11] Eugenio Moggi. Computational Lambda-calculus and Monads. Report ECS-LFCS-88-66, University of Edinburgh, Edinburgh, Scotland, October 1988.

[12] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89)*, pages 14–23. IEEE Computer Society, 1989.

[13] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[14] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[15] Gordon D. Plotkin and John Power. Notions of computation determine monads. In *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2002.

[16] Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.

[17] Amr Sabry and Philip Wadler. A reflection on call-by-value. In Robert Harper and Richard L. Wexelblat, editors, *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming, ICFP 1996, Philadelphia, Pennsylvania, USA, May 24-26, 1996*, pages 13–24. ACM, 1996.

[18] Alex Simpson. Reduction in a linear lambda-calculus with applications to operational semantics. In Jürgen Giesl, editor, *Term Rewriting and Applications*, pages 219–234, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[19] Vincent van Oostrom. Private communication via electronic mail, 2020.

[20] Philip Wadler and Peter Thiemann. The marriage of effects and monads. *ACM Trans. Comput. Log.*, 4(1):1–32, 2003.

A Confluent Trace Semantics for Probabilistic Lambda Calculus

Andrew Kenyon-Roberts

***Abstract*—Probabilistic lambda calculus has a weaker version of the confluence property of plain lambda calculus. However, in the usual formulation, this only applies in the sense of distributions, but not to individual traces of random choices. A new labelling scheme for random choices is introduced for PPCF, a simply-typed functional language with explicit recursion, real numbers and a random sampling operator, that allows a confluent version of trace semantics to be defined (with a restricted set of call-by-value reduction strategies).**

I. Introduction

Non-probabilistic lambda calculi are generally confluent, i.e. if a term A reduces to both B_1 and B_2, there is some C to which both B_1 and B_2 reduce, so the reduction order mostly doesn't matter. In the probabilistic case, this may not be true, because β-reduction can duplicate samples, so the outputs of the copies of the sample may be identical or independent, depending on whether the sample is taken before or after β-reduction. Consider for example the term $(\lambda x.x+x)\,\mathsf{sample}$, where sample reduces to a number chosen uniformly at random from the interval $I = [0, 1]$. If it is reduced in call-by-value order, first the sample reduces to some number r, then the β redex is evaluated, then r is added to itself, yielding $\underline{2r}$. If it is reduced in call-by-name order instead, first the β redex is reduced, yielding $\mathsf{sample} + \mathsf{sample}$, then the samples are evaluated independently and added, yielding $\underline{r + r'}$. As r and r' are independent, the distribution of results is triangular, with support $[0, 2]$ and peak at 1, which is different from the uniform distribution of results in the CbV case.

The results obtained by CbV and CbN evaluation differ in a significant way, however, there are some cases where the order of evaluation doesn't matter. For example, in $\mathsf{sample}+\mathsf{sample}$, the order in which the samples are evaluated doesn't affect the final result, and in $(\lambda x.\mathsf{sample})\,\underline{0}$, the β redex and the sample can be evaluated in either order. In order to obtain the desired confluence result, we restrict our attention to a class of reduction strategies that are equivalent to CbV, as the CbN semantics is less expressive, being unable to force evaluation of a random choice and duplicate the result.

Even with such a restriction, a trace semantics in the usual style would not be entirely confluent. In the normal sort of trace semantics [1], there is a sequence of samples, the *trace*, selected at random from a trace space such as $I^{\mathbb{N}}$, then for every sample statement reduction, the next sample from the trace is used in order, so that the samples in the trace are effectively each labelled by a number corresponding to the execution order of the sample statements. Consider the evaluation of the term $\mathsf{sample} - \mathsf{sample}$ using one of these simple linear traces, $(1, 0, \dots)$. It would reduce to either 1 or -1 depending on the order of evaluation of the samples, as that determines which sample from the pre-selected sequence is used for each one. To fix this, rather than pre-selecting samples according to the order they'll be drawn in, they can be labelled according to the position in the term where they'll be used instead.

Further details, including all of the missing proofs, can be found in [2, §IV, §D].

A. Outline

First, the syntax of the language PPCF is introduced. Positions are defined as a way of addressing sample statements within a program independently of the reduction order. Next, a version of the reduction relation is presented that is nondeterministic, so that it allows a choice of what order to perform reductions in. The notion of positions is extended to potential positions, for samples which may appear later in the reduction sequence but not necessarily in the initial term. In order to allow potential positions in different reduction sequences to be considered equivalent, a relation $\sim^*$ is defined, and finally, all of these are used to construct a confluent version of the trace semantics, $\Rightarrow$, that is still nondeterministic in reduction order, but does specify the outcome of random choices.

II. Syntax of Probabilistic PCF

The language PPCF is a call-by-value version of PCF with sampling of real numbers from the closed interval $[0, 1]$ [3–5]. Types and terms are defined as follows, where r is a real number, x is a variable, $f : \mathbb{R}^n \to \mathbb{R}$ is any measurable function, and Γ is an environment:

$$\begin{aligned} \textit{types}\ & A, B ::= \mathsf{R} \mid A \to B \\ \textit{values}\ & V ::= \lambda x.M \mid \underline{r} \\ \textit{terms}\ & M, N ::= V \mid x \mid M_1\, M_2 \mid \underline{f}(M_1, \dots, M_n) \mid \mathsf{Y}\, M \\ & \qquad \mid \mathsf{if}(M < 0, N_1, N_2) \mid \mathsf{sample} \end{aligned}$$

The typing rules are standard (see Fig. 1). The restriction to well-typed terms is only necessary here in order to avoid reaching terms which contain nonsense such as applying a number as though it were a function, so a more liberal type system would work just as well. Simple types are just used for simplicity. Terms are identified up to α-equivalence, as usual. The set of all terms is denoted Λ, and the set of closed terms is denoted Λ^0.

$$\frac{}{\Gamma; x : A \vdash x : A} \qquad \frac{\Gamma; x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B}$$

$$\frac{\Gamma \vdash M : (A \to B) \to (A \to B)}{\Gamma \vdash \mathsf{Y}M : (A \to B)} \qquad \frac{\Gamma \vdash M : \mathsf{R} \quad \Gamma \vdash N_1 : A \quad \Gamma \vdash N_2 : A}{\Gamma \vdash \mathsf{if}(M < 0, N_1, N_2) : A}$$

$$\frac{}{\underline{r} : \mathsf{R}} \qquad \frac{}{\Gamma \vdash \mathsf{sample} : \mathsf{R}} \qquad \frac{\Gamma \vdash M_1 : \mathsf{R} \quad \dots \quad \Gamma \vdash M_n : \mathsf{R}}{\Gamma \vdash \underline{f}(M_1, \dots, M_n) : \mathsf{R}}\ (f : \mathbb{R}^n \to \mathbb{R})$$

Figure 1. Typing rules of PPCF

III. Positions

A *position* is a finite sequence of steps into a term, defined inductively as

$$\begin{aligned}\alpha ::= \cdot \mid \lambda;\alpha \mid @_1;\alpha \mid @_2;\alpha \mid \underline{f}_i;\alpha \\ \mid \mathsf{Y};\alpha \mid \mathsf{if}_1;\alpha \mid \mathsf{if}_2;\alpha \mid \mathsf{if}_3;\alpha.\end{aligned}$$

The *subterm of* M *at* α, denoted $M \mid \alpha$, is defined as

$$\begin{aligned} M \mid \cdot &= M \\ \lambda x.M \mid \lambda;\alpha &= M \mid \alpha \\ M_1\,M_2 \mid @_i;\alpha &= M_i \mid \alpha \quad \text{for } i = 1,2 \\ \underline{f}(M_1,\dots,M_n) \mid \underline{f}_i;\alpha &= M_i \mid \alpha \quad \text{for } i \le n \\ \mathsf{Y}M \mid \mathsf{Y};\alpha &= M \mid \alpha \\ \mathsf{if}(M_1 < 0, M_2, M_3) \mid \mathsf{if}_i;\alpha &= M_i \mid \alpha \quad \text{for } i = 1,2,3 \end{aligned}$$

so that every subterm is located at a unique position, but not every position corresponds to a subterm (e.g. $x\,y \mid \lambda$ is undefined). A position such that $M \mid \alpha$ does exist is said to *occur* in M. *Substitution* of N at position α in M, written $M[N/\alpha]$, is defined similarly. For example, let $M = \lambda x\,y.y\,(\mathsf{if}(x < 0, y\,(\underline{f}(x)), \underline{3}))$ and $\alpha = \lambda;\lambda;@_2;\mathsf{if}_2;@_2$ then $M[\mathsf{sample}/\alpha] = \lambda x\,y.y\,(\mathsf{if}(x < 0, y\,\mathsf{sample}, \underline{3}))$.

Two subterms N_1 and N_2 of a term M, corresponding to positions α_1 and α_2, can overlap in a few different ways. If α_1 is a prefix of α_2 (written as $\alpha_1 \le \alpha_2$), then N_2 is also a subterm of N_1. If neither $\alpha_1 \le \alpha_2$ nor $\alpha_1 \ge \alpha_2$, the positions are said to be *disjoint*. The notion of disjointness is mostly relevant in that if α_1 and α_2 are disjoint, performing a substitution at α_1 will leave the subterm at α_2 unaffected.

Thus we can define a nondeterministic reduction relation $\to$.

Definition III.1. The binary relation $\to$ is defined by the following rules, each is conditional on a redex occurring at position α in the term M:

$$\begin{aligned} \text{if } M \mid \alpha = (\lambda x.N)V,\ M &\to M[N[V/x]/\alpha] \\ \text{if } M \mid \alpha = \underline{f}(\underline{r}_1,\dots,\underline{r}_n),\ M &\to M[\underline{f(r_1,\dots,r_n)}/\alpha] \\ \text{if } M \mid \alpha = \mathsf{Y}\lambda x.N,\ M &\to M[\lambda z.N[(\mathsf{Y}\lambda x.N)/x]z/\alpha] \\ &\text{where } z \text{ is not free in } N \end{aligned}$$

if $M \mid \alpha = \mathsf{if}(\underline{r} < 0, N_1, N_2)$, $M \to M[N_1/\alpha]$ where $r < 0$

if $M \mid \alpha = \mathsf{if}(\underline{r} < 0, N_1, N_2)$, $M \to M[N_2/\alpha]$ where $r \ge 0$

if $M \mid \alpha = \mathsf{sample}$ and λ does not occur after $@_2$ or Y in α, $M \to M[\underline{r}/\alpha]$ where $r \in [0,1]$.

In each of these cases, $M \mid \alpha$ is the *redex*, and the reduction *takes place at* α. Each subterm can be a redex in at most one way, but there can be multiple redexes at different positions.

The argument of a β redex and the body of a Y redex may be duplicated by those reductions. It is therefore these cases that need to be handled carefully to avoid duplicating samples at the wrong time. In both cases, the potentially duplicated part must already be a value, which excludes terms like sample or $\mathsf{sample}+\underline{1}$, which should be evaluated before being duplicated. In the other direction, if a sample occurs inside of a λ, it may need to be duplicated before being evaluated, which is why a sample reduction isn't allowed inside a λ inside a Y or the right side of an application. These restrictions are in some cases unnecessarily strict, for example, in $(\lambda x.x)((\lambda y.\mathsf{sample})\underline{0})$, it would be fine to evaluate the sample first, but they are at least sufficient to ensure confluence in terms of the distribution of results. Getting individual traces to behave correctly will take more work though.

IV. Skeletal Reduction Sequences

Labelling the pre-chosen samples by the positions in the term by using $I^{\{\alpha \mid (M \mid \alpha) = \mathsf{sample}\}}$ as the trace space would not be sufficient to solve the issue of different samples being used in corresponding locations in different reduction sequences because in some cases, a sample will be duplicated before being reduced, for example, in $(\lambda x.x\,\underline{0} + x\,\underline{0})(\lambda y.\mathsf{sample})$, both of the sample redexes that eventually occur originate at $@_2;\lambda$. It is therefore necessary to consider possible positions that may occur in other terms reachable from the original term. Even this is itself inadequate because some of the positions

in different reachable terms need to be considered the same, and the number of reachable terms is in general uncountable, which leads to measure-theoretic issues.

We are thus led to consider the reduction relation on skeletons. Define a *skeleton* to be a term but, instead of having real constants $\underline{r}$, it has a placeholder X, so that each term M has a skeleton $Sk(M)$, and each skeleton S can be converted to a term $S[r]$ given a vector r of n real numbers to substitute in, where n is the number of occurrences of X in S. Positions in a skeleton and the reduction relation $\to$ on skeletons can be extended from the definitions on terms in the obvious way, with $\mathsf{if}(\mathsf{X} < 0, A, B)$ reducing nondeterministically to both A and B, sample reducing to X, and X considered as (the skeletal equivalent of) a value, so that $(\lambda x.A)\,\mathsf{X}$ reduces to $A[\mathsf{X}/x]$. For example, we have $(\lambda x.\mathsf{if}(x < 0, x, \mathsf{X}))\,\mathsf{sample} \to (\lambda x.\mathsf{if}(x < 0, x, \mathsf{X}))\,\mathsf{X} \to \mathsf{if}(\mathsf{X} < 0, \mathsf{X}, \mathsf{X}) \to \mathsf{X}$.

Given a closed term M, let $L_0(M)$ be the set of pairs, the first element of which is a $\to$-reduction sequence of skeletons starting at $Sk(M)$, and the second of which is a position in the final skeleton of the reduction sequence. As with the traces from $I^{\mathbb{N}}$ used to pre-select samples to use in the standard trace semantics, modified traces, which are elements of $I^{L_0(M)}$ (with one more caveat introduced after Def. V.2), will be used to pre-select a sample from I for each element of $L_0(M)$, which will then be used if a sample reduction is ever performed at that position.

A (skeletal) reduction sequence is assumed to contain the information on the locations of all of the redexes as well as the actual sequence of skeletons that occurs. For example, $(\lambda x.x)((\lambda x.x)\,\mathsf{X})$ could reduce to $(\lambda x.x)\,\mathsf{X}$ with the redex at either $\cdot$ or $@_2$, and these give different reduction sequences.

Example IV.1. Consider the terms

$$A[M] = \mathsf{if}(\mathsf{if}(M > 0, I, I)(\lambda y.\mathsf{sample})\,\underline{0} - \underline{0.5} > 0, \underline{0}, \Omega)$$
$$B = \mathsf{if}(\mathsf{sample} - \underline{0.5} > 0, \underline{0}, \Omega)$$

If terms rather than skeletons were used to label samples, the set of modified traces where $A[\mathsf{sample}]$ terminates would be

$$\bigcup_{r\in[0,1]} \{s \mid s(A[\mathsf{sample}], \mathsf{if}_1; \underline{=}_1; @_1; @_1; \mathsf{if}_1) = r, \; s(A[\mathsf{sample}] \to A[\underline{r}] \to^* B, \mathsf{if}_1; \underline{=}_1) > 0.5\}.$$

This is a rather unwieldy expression, but the crucial part is that r occurs twice in the conditions on s: once as the value a sample must take, and once in the location of a sample. As this set is unmeasurable, the termination probability would not even be well-defined. Labelling samples by skeletons instead, this problem does not occur because there are only countably many skeleton, and at each step in a reduction sequence, only finitely many could have occurred yet. Although skeletal reduction sequences omit the information on what the results of sampling were, they still contain all the necessary information on how many, and which, reductions took place.

For this particular term, $Sk(A[\underline{r}])$ does not depend on the value of r, therefore the set where it terminates becomes simply the following, which is measurable.

$$\{s \mid s(Sk(A[\mathsf{sample}]) \to Sk(A[\underline{0}]) \to^* Sk(B), \mathsf{if}_1; \underline{=}_1) > 0.5\}$$

Reduction sequences are used rather than reachable skeletons because if the same skeleton is reached twice, different samples may be needed:

Example IV.2. Consider the term $M = \mathsf{Y}(\lambda f x.\,\mathsf{if}(\mathsf{sample} - \underline{0.5} < 0, f\,x, x))\,\underline{0}$, which reduces after a few steps to $N = \mathsf{if}(\mathsf{sample} - \underline{0.5} < 0, M, \underline{0})$. If we label samples by just skeletons and positions, and the pre-selected sample for $(Sk(N), \mathsf{if}_1; \underline{=}_1)$ is less than 0.5, N reduces back to M, then N again, then the same sample is used the next time, therefore it's an infinite loop, whereas if samples are labelled by reduction sequences, the samples for $M \to^* N$ are independent from the samples for $M \to^* N \to M \to^* N$, and so on.

The reduction sequences of skeletons will often be discussed as though they were just skeletons, identifying them with their final skeletons. With this abuse of notation, a reduction sequence N (actually $N_1 \to^* N_n = N$) may be said to reduce to a reduction sequence O, where the reduction sequence implicitly associated with the final skeleton O is $N_1 \to^* N_n \to O$.

V. Potential Positions

This is still not quite sufficient to attain confluence because sometimes the same samples must be used at corresponding positions in different reduction sequences.

Example V.1. The term $M = \mathsf{sample} + \mathsf{sample}$ has the reachable skeletons $N_1 = \mathsf{X} + \mathsf{sample}$, $N_2 = \mathsf{sample} + \mathsf{X}$, $O = \mathsf{X} + \mathsf{X}$ and X, with reductions $M \to N_1 \to O \to \mathsf{X}$ and $M \to N_2 \to O \to \mathsf{X}$. In the reduction $M \to N_1$, the sample labelled $(M, \underline{+}_1)$ is used, and in the reduction $N_2 \to O$, the sample labelled $(M \to N_2, \underline{+}_1)$ is used. Each of these samples becomes the value of the first numeral in O in their respective reduction sequences, therefore in order for confluence to be attained, they must be the same. Which elements of $L_0(M)$ must match can be described by the relation $\sim^*$:

Definition V.2. The relation $\sim$ is defined as the union of the minimal symmetric relations $\sim_p$ ("p" for parent-child) and $\sim_c$ ("c" for cousin) satisfying

(i) If N reduces to O with the redex at position α, and β is a position in N disjoint from α, then $(N, \beta) \sim_p (O, \beta)$.

(ii) If N β-reduces to O at position α, β is a position in $N \mid \alpha; @_1; \lambda$ and $N \mid \alpha; @_1; \lambda; \beta$ is not the variable involved in the reduction, $(N, \alpha; @_1; \lambda; \beta) \sim_p (O, \alpha; \beta)$.

(iii) If N if-reduces to O at position α, with the first resp. second branch being taken, and $\alpha; \mathsf{if}_i; \beta$ occurs in N (where $i = 2$ resp. 3), $(N, \alpha; \mathsf{if}_i; \beta) \sim_p (O, \alpha; \beta)$.

(iv) If N, O_1 and O_2 match any of the following cases:

a) N contains redexes at disjoint positions α_1 and α_2, O_1 is N reduced first at α_1 then α_2 and O_2 is N reduced first at α_2 then at α_1.

b) $N \mid \alpha = \mathsf{if}(\underline{r} < 0, N_1, N_2)$, where $r < 0$ (or, respectively, $r \geq 0$), $(N_2$ resp. $N_1) \mid \beta$ is a redex, and O_1 is N reduced at α and O_2 is N reduced first at $\alpha; (\mathsf{if}_3$ resp. $\mathsf{if}_2); \beta$ then at α.
c) $N \mid \alpha = \mathsf{if}(\underline{r} < 0, N_1, N_2)$, where $r < 0$ (or, respectively, $r \geq 0$), $(N_1$ resp. $N_2) \mid \beta$ is a redex, and O_1 is N reduced first at α then at $\alpha; \beta$ and O_2 is N reduced first at $\alpha; (\mathsf{if}_2$ resp. $\mathsf{if}_3); \beta$ then at α.
d) $N \mid \alpha = (\lambda x.A)B$, there is a redex in A at position β, O_1 is N reduced first at α then at $\alpha; \beta$, and O_2 is N reduced first at $\alpha; @_1; \lambda; \beta$ then at α.
e) $N \mid \alpha = (\lambda x.A)B$, $B \mid \beta$ is a redex, $(\gamma_i)_i$ is a list of all the positions in A where $A \mid \gamma = x$, ordered from left to right, O_1 is N reduced first at $\alpha; @_2; \beta$ then at α, and O_2 is N reduced first at α then at $\alpha; \gamma_i; \beta$ for each i in order.
f) $N \mid \alpha = \mathsf{Y}(\lambda x.A)$, A reduced at β is A', $(\gamma_i)_i$ is a list of all the positions where $A' \mid \gamma = x$, ordered from left to right, O_1 is N reduced first at $\alpha; \mathsf{Y}; \lambda; \beta$ then at α, and O_2 is N reduced first at α then at $\alpha; \lambda; @_1; \gamma_i; Y; \lambda; \beta$ for each i in order where γ_i is left of β then at $\alpha; \lambda; @_1; \beta$ then at $\alpha; \lambda; @_1; \gamma_i; Y; \lambda; \beta$ for the remaining values of i.

(in which case O_1 and O_2 are equal as skeletons, but with different reduction sequences), O_1' and O_2' are the results of applying some reduction sequence to each of O_1 and O_2 (the same reductions in each case, which is always possible because they're equal skeletons), and δ is a position in O_1' (or equivalently O_2'), then $(O_1', \delta) \sim_c (O_2', \delta)$.

Example V.3. In Ex. V.1, $(M, \underline{+}_1) \sim_p (M \to N_2, \underline{+}_1)$ by case i of $\sim_p$ (because the reduction $M \to N_2$ occurs at $\underline{+}_2$, which is disjoint from $\underline{+}_1$), and similarly, $(M, \underline{+}_2) \sim_p (M \to N_1, \underline{+}_2)$.

If we extend it to have three samples, $\sim_c$ becomes necessary as well: Let $M_{sss} = \mathsf{sample} + \mathsf{sample} + \mathsf{sample}$ (taking the three-way addition to be a single primitive function), $M_{Xss} = \mathsf{X} + \mathsf{sample} + \mathsf{sample}$, and so on. There are then reduction sequences $M_{sss} \to M_{Xss} \to M_{XXs} \to M_{XXX} \to \mathsf{X}$ and $M_{sss} \to M_{sXs} \to M_{XXs} \to M_{XXX} \to \mathsf{X}$. For the first two reductions, these reduction sequences take the same samples by $\sim_p$, case i, as in Ex. V.1 . The next reduction uses the samples labelled by $(M_{sss} \to M_{Xss} \to M_{XXs}, \underline{+}_3)$ and $(M_{sss} \to M_{sXs} \to M_{XXs}, \underline{+}_3)$, which are related by $\sim_c$, case a, therefore when these reduction sequences reach M_{XXX}, they still contain all the same numbers, as desired.

The reflexive transitive closure $\sim^*$ of this relation is used to define the set of *potential positions* $L(M) = L_0(M)/\sim^*$, and each equivalence class can be considered as the same position as it may occur across multiple reachable skeletons. If $(N, \alpha) \sim^* (O, \beta)$, then $N \mid \alpha$ and $O \mid \beta$ both have the same shape (i.e. they're either both the placeholder X, both variables, both applications, both samples etc.), therefore it's well-defined to talk of the set of *potential positions where there is a* sample, $L_s(M)$. The new sample space is then defined as $I^{L_s(M)}$, with the Borel σ-algebra and product measure. Since $I^{L_s(M)}$ is a countable product, the measure space is well-defined [6, Cor. 2.7.3].

VI. The Confluent Trace Semantics

Before defining the new version of the reduction relation, the following lemma is necessary for it to be well-defined.

Lemma VI.1. *The relation $\sim$ is defined on $L_0(M)$ with reference to a particular starting term M, so different versions, $\sim_M$ and $\sim_N$, can be defined starting at different terms. If $M \to N$, then $\sim_N^*$ is equal to the restriction of $\sim_M^*$ to $L_0(N)$.*

At each reduction step $M \to N$, the sample space must be restricted from $I^{L_s(M)}$ to $I^{L_s(N)}$. The injection $L_0(N) \to L_0(M)$ is trivial to define by appending $Sk(M) \to Sk(N)$ to each path, and using Lem. VI.1, this induces a corresponding injection on the quotient, $L(N) \to L(M)$. The corresponding map $L_s(N) \to L_s(M)$ is then denoted $i(M \to N)$.

Definition VI.2 ($\Rightarrow$ reduction)**.** This version of the reduction relation now specifies the results of sample reductions, but is still nondeterministic with respect to the order of reduction. It relates $\biguplus_{M \in \Lambda_0} I^{L_s(M)}$ to itself. We write an element of $\biguplus_{M \in \Lambda_0} I^{L_s(M)}$ as (M', s) where the term $M' \in \Lambda^0$ and $s \in I^{L_s(M')}$.

$$(M, s) \Rightarrow (N, s \circ i(M \to N)) \text{ if } M \to N \text{ at } \alpha \text{ and either}$$
$$\text{the redex is not sample, or}$$
$$M \mid \alpha = \mathsf{sample} \text{ and } N = M[\underline{s(Sk(M), \alpha)}/\alpha]$$

This reduction relation now has all of the properties required of it. In particular, it can be considered an extension of the standard trace semantics (as will be seen later in Thm. VI.5), and also:

Lemma VI.3. *The relation $\Rightarrow$ is confluent.*

In order to show that $\Rightarrow$ behaves as expected, the following lemma is also necessary, in order to show that a sample is never used multiple times in the same reduction sequence:

Lemma VI.4. *If $M \to N$, with the redex at position α, then no position in any term reachable from N is related by $\sim^*$ to (M, α).*

The reduction relation $\Rightarrow$ is nondeterministic, so it admits multiple possible reduction strategies. A *reduction strategy* starting from a closed term M is a measurable partial function f from $Rch(M)$ to positions, such that for any reachable term N where f is defined, $f(N)$ is a position of a redex in N, and if $f(N)$ is not defined, N is a value. Using a reduction strategy f, a subset of $\Rightarrow$ that isn't nondeterministic, $\Rightarrow_f$, can be defined by $(N, s) \Rightarrow_f (N', s')$ just if $(N, s) \Rightarrow (N', s')$ and N reduces to N' with the redex at $f(N)$.

The usual call-by-value semantics can be implemented as one of these reduction strategies, given by (with V a value

and T a term that isn't a value and M a general term)

$$\begin{aligned}
\mathrm{cbv}(TM) &= @_1; \mathrm{cbv}(T) \\
\mathrm{cbv}(VT) &= @_2; \mathrm{cbv}(T) \\
\mathrm{cbv}(\underline{f}(V_1, \ldots, V_{k-1}, T, M_{k+1}, \ldots, M_n)) &= \underline{f}_k; \mathrm{cbv}(T) \\
\mathrm{cbv}(\mathsf{Y}T) &= \mathsf{Y}; \mathrm{cbv}(T) \\
\mathrm{cbv}(\mathsf{if}(T < 0, M_1, M_2)) &= \mathsf{if}_1; \mathrm{cbv}(T) \\
\mathrm{cbv}(V) &\text{ is undefined} \\
\mathrm{cbv}(T) &= \cdot \text{ otherwise}
\end{aligned}$$

(this last case covers redexes at the root position).

A closed term M terminates with a given reduction strategy f and samples s if there is some natural number n such that $(M, s) \Rightarrow_f^n (N, s')$ where f gives no reduction at N. The term is *almost surely terminating (AST) w.r.t.* f if it terminates with f for almost all s.

This reduction strategy allows the confluent trace semantics to be related to the standard version of the trace semantics with a fixed reduction order and linear traces. In [2], which gives the full definition of the standard trace semantics, this is used to prove the following theorems that allow termination results to be transferred from the confluent trace semantics to the standard trace semantics.

Theorem VI.5. *A closed term M is AST with respect to* cbv *iff it is AST.*

Theorem VI.6. *If M terminates with some reduction strategy f and trace s, it terminates with* cbv *and s.*

Corollary VI.7 (Reduction strategy independence)**.** *If M is AST with respect to any reduction strategy, it is AST.*

Proof. Suppose M is AST w.r.t. f. Let the set of samples with which it terminates with this reduction strategy be X. By Thm. VI.6, M also terminates with cbv and every element of X, and X has measure 1, by assumption, therefore M is AST with respect to cbv therefore by Thm. VI.5 it is AST. □

References

[1] J. Borgström, U. Dal Lago, A. D. Gordon, and M. Szymczak, "A lambda-calculus foundation for universal probabilistic programming," *ACM SIGPLAN Notices*, vol. 51, no. 9, pp. 33–46, 2016.

[2] A. Kenyon-Roberts and L. Ong, "Supermartingales, ranking functions and probabilistic lambda calculus," *arXiv preprint arXiv:2102.11164*, 2021.

[3] T. Ehrhard, M. Pagani, and C. Tasson, "Full Abstraction for Probabilistic PCF," *Journal of the ACM*, vol. 65, no. 4, pp. 1–44, apr 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3208081.3164540

[4] ——, "Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming," *PACMPL*, vol. 2, no. POPL, pp. 59:1–59:28, 2018. [Online]. Available: http://doi.acm.org/10.1145/3158147

[5] C. Mak, C.-H. L. Ong, H. Paquet, and D. Wagner, "Densities of almost surely terminating probabilistic programs are differentiable almost everywhere," in *ESOP 2021*, 2021, to appear. https://arxiv.org/abs/2004.03924.

[6] R. B. Ash and C. Doléans-Dade, *Probability and measure Theory*. Harcourt Academic Press, 2000.

Confluence in string rewriting systems compatible with a crystal structure

Uran Meha

Université de Lyon 1
meha@math.univ-lyon1.fr

Abstract

A crystal is a kind of directed labeled graph arising in the field of representation theory. We consider an adapted abstract notion of crystals, called $\mathcal{K}$-graphs. A $\mathcal{K}$-graph structure on a free monoid is a directed colored graph structure satisfying certain conditions for the product of the monoid. A $\mathcal{K}$-congruence on the free monoid is one that identifies isomorphic connected components of the $\mathcal{K}$-graph. In this work we define a notion of $\mathcal{K}$-string rewriting systems which generate such $\mathcal{K}$-congruences. For a class of $\mathcal{K}$-graphs called proper, the interaction of the string rewriting system with the $\mathcal{K}$-graph structure reduces the proofs of the rewriting properties of termination and (local) confluence to a family of reduced words in the free monoid, called words of highest weight. From this we deduce $\mathcal{K}$-versions of Newman's lemma, critical pair lemma, and Squier's coherent completion theorem. Finally we illustrate these constructions with an example for the plactic monoid of type A. The constructions in this work are phrased in terms of $\mathcal{K}$-graphs, though many of their applications lie in the original context of crystals.

1 Introduction

A central approach in the study of confluence in rewriting theory is to reduce the problem to a subset of branchings namely to local-confluence, and to critical branchings. This approach is detailed in two results: Newman's Lemma [8], where the property of confluence of a terminating string rewriting system is equated to the property of local confluence; and the Critical Pair Lemma (CPL) [6], where the local confluence of a string rewriting system is equated to confluence of its critical branchings, which are pairs of overlapping rules on a minimal source.

In an algebraic context, rewriting theory has found applications in the higher dimensional study of objects like monoids, small categories, and algebras over a field. This consists of realizing the object in question by a presentation with generators and oriented relations compatible with the defining axioms of the algebraic object. In this context, a confluence diagram may be regarded as a *relation* between two rewriting sequences, which in turn are relations in the presentation itself, thus one may view the confluence diagrams as *relations between relations*. A question in this direction is how to obtain a set of generating relations between relations from the given presentation, so that any confluence diagram of the rewriting system can be described in terms of this specified set. This question is answered by Squier in [10] in the context of monoids and small categories. Namely, given a presentation of a monoid or a small category by generators and oriented relations, the rewriting system of which is convergent, then the generating relations between relations are the confluence diagrams of critical branchings. The data of generators, generating relations, and generating relations between relations is called a *coherent presentation*. An important aspect of the study of relations between relations in presentations of monoids, is that it provides an algorithmic approach to the study of the monoid's lower-dimensional homology. Note however that a critical branching may admit several confluence diagrams in Squier's construction. To complete this algorithmic point of view, Malbos

and Guiraud in [3] employ a notion of a *normalization strategy* which is a deterministic way to choose confluence diagrams for critical branchings. In their work, they take the ideas of Squier even further: they show that for a small category presented by generators and oriented relations, whose string rewriting system is convergent, one can construct a cofibrant replacement for the category. In a grander scheme these constructions facilitate an algorithmic approach to the study of the homology of small categories.

The works of Squier, and Guiraud and Malbos provide a way of determining coherent presentations of small categories and monoids from convergent ones. Finding a convergent presentation in the first place and computing the confluence diagrams of critical branchings remains a difficult task, as in general this problem is heavily dependent on the intrinsic properties of the monoid and the presentation.

In this work, we consider a notion of a string rewriting system adapted to the theory of *crystals*, and give corresponding versions of three classical results in rewriting theory. The notion of crystals was first defined by Kashiwara in [5] in his study of representations of complex semisimple Lie algebras. We phrase our constructions in terms of an abstracted version of crystals called $\mathcal{K}$*-graphs*, which have been adapted from [1]. This work emerged from a study of coherence of the plactic monoid of type C in [7], and forms part of a forthcoming PhD thesis by the author.

In Section 2 we introduce the notion of a $\mathcal{K}$-graph as a directed colored graph satisfying certain conditions. We then consider a $\mathcal{K}$-graph structure on the free monoid generated by the vertices of a $\mathcal{K}$-graph. A $\mathcal{K}$*-congruence* is one that identifies isomorphic connected components of the $\mathcal{K}$-graph on the free monoid. We then introduce a notion of a $\mathcal{K}$-string rewriting system. This is a string rewriting system that is compatible with the $\mathcal{K}$-graph structure, and such that the congruence generated by it is a $\mathcal{K}$-congruence. In Section 3 we show that if $\mathcal{K}$ is *proper*, then the study of rewriting properties of termination, and local confluence is reduced to a subfamily of words called *words of highest weight*. In particular we obtain $\mathcal{K}$-versions of Newman's lemma, the critical pair lemma, and of Squier's coherent completion theorem. In Appendix A we illustrate these constructions and results with an example of the *plactic monoid of type A*. Finally in Section 4 we briefly discuss how this approach could be extended to higher dimensions in accordance with the work of Guiraud and Malbos [3].

2 $\mathcal{K}$-string rewriting systems

A $\mathcal{K}$*-graph* is a directed colored graph Γ with vertex set $V(\Gamma)$, and with edges colored from a set I, satisfying the following conditions

(P1) for any $x \in V(\Gamma)$ and $i \in I$, there exists at most one edge e with *source (target)* x and color i,

(P2) for any $i \in I$, there exists no infinite directed path in Γ with edges colored by i.

It is practical to realize the $\mathcal{K}$-graph structure via the *Kashiwara operators*, which are partial maps e_i and f_i on $V(\Gamma)$ defined by setting

$$x \xrightarrow{i} y \quad \text{if and only if} \quad y = f_i.x, \text{ and } x = e_i.y.$$

Remark 2.1. *The notion of crystals was introduced by Kashiwara in [5] in his study of the representation theory of quantum groups. In this work the constructions are phrased in terms of $\mathcal{K}$-graphs, which are an abstract graph-theoretic adaptation of crystals as introduced in [1]. We remark that a large class of crystals satisfies* **(P1), (P2)**.

Given a $\mathcal{K}$-graph Γ, the graph structure extends to the free monoid on the vertices $V(\Gamma)$ of Γ, denoted Γ^*. The Kashiwara operators e_i and f_i extend to Γ^* inductively on the lengths of words $w = uv \in \Gamma^*$ as follows

$$e_i.(uv) = \begin{cases} (e_i.u)v & \text{if } \varphi_i(u) \geq \varepsilon_i(v), \\ u(e_i.v) & \text{if } \varphi_i(u) < \varepsilon_i(v), \end{cases} \tag{1}$$

and

$$f_i.(uv) = \begin{cases} (f_i.u)v & \text{if } \varphi_i(u) > \varepsilon_i(v), \\ u(f_i.v) & \text{if } \varphi_i(u) \leq \varepsilon_i(v), \end{cases} \tag{2}$$

where $\varepsilon_i, \varphi_i : \Gamma^* \longrightarrow \mathbb{N}$ are also defined inductively via

$$\varepsilon_i(w) = \#\{e_i.w, e_i^2.w, \dots\}, \quad \varphi_i(w) = \#\{f_i.w, f_i^2.w, \dots\},$$

which are finite quantities by an iteration of **(P2)**. We remark here a few consequences of these definitions:

i) e_i and f_i are partial operators on Γ^*: e.g. if $\varphi_i(u) \geq \varepsilon_i(v)$ and $e_i.u$ is undefined, then $e_i.(uv)$ is also undefined;

ii) the definition of e_i and f_i on a word w is independent of the factorization $w = uv$;

iii) e_i and f_i are inverse operators: $f_i.(e_i.w) = w$ and $e_i.(f_i.w) = w$.

Thus the free monoid Γ^* carries a directed colored graph structure, and we call it the *free $\mathcal{K}$-monoid generated by* Γ. As Γ^* is a graph, we have a notion of *connected components* in Γ^*. The connected component of $w \in \Gamma^*$ is denoted by $\mathbf{B}(w)$. Using this notion, we specify a type of congruence on the free $\mathcal{K}$-monoid.

Definition 2.2. *Let Γ be a $\mathcal{K}$-graph, and Γ^* the corresponding free $\mathcal{K}$-monoid. A $\mathcal{K}$-congruence on Γ^* is a congruence $\sim$ such that if $w \sim w'$, then*

i) *there exists a directed colored graph isomorphism $p : \mathbf{B}(w) \longrightarrow \mathbf{B}(w')$ such that $p(w) = w'$,*

ii) *if $e_i.w$ (resp. $f_i.w$) is defined, then so is $e_i.w'$ (resp. $f_i.w'$) and we have*

$$e_i.w \sim e_i.w' \qquad (\text{resp. } f_i.w \sim f_i.w').$$

The largest such congruence, denoted $\sim_\Gamma$ is defined by setting $w \sim_\Gamma w'$ if and only if there exists an isomorphism p as in Definition 2.2 **i)**.

We specify here a class of $\mathcal{K}$-graphs that occurs often and has practical combinatorial advantages. A word $w \in \Gamma^*$ is called *a word of highest weight* if $e_i.w$ is undefined for all $i \in I$. If Γ is such that every connected component $\mathbf{B}(w) \subset \Gamma^*$ contains a unique word of highest weight, the $\mathcal{K}$-graph Γ is called *proper*.

Next we introduce a notion of string rewriting which is compatible with a $\mathcal{K}$-graph structure. One may view the next definition simply as an oriented generating data for a $\mathcal{K}$-congruence, hence the similarity with Definition 2.2.

Definition 2.3. *A $\mathcal{K}$-string rewriting system is a string rewriting system (Γ^*, R) where Γ is a $\mathcal{K}$-graph, and such that if $w \Longrightarrow w'$ is a rewriting rule in R, then*

i) *there exists a directed colored graph isomorphism $p : \mathbf{B}(w) \longrightarrow \mathbf{B}(w')$ such that $p(w) = w'$,*

ii) *if $e_i.w$ (resp. $f_i.w$) is defined, then so is $e_i.w'$ (resp. $f_i.w'$) and we have*

$$(e_i.w \Longrightarrow e_i.w') \in R \qquad (\textit{resp. } (f_i.w \Longrightarrow f_i.w') \in R).$$

For a $\mathcal{K}$-string rewriting system (Γ^*, R), the congruence in Γ^* generated by R is a $\mathcal{K}$-congruence. Thus $\mathcal{K}$-string rewriting systems are well-adapted at studying such congruences. We call a $\mathcal{K}$-string rewriting system *proper* if Γ^* is proper.

3 Confluence for $\mathcal{K}$-string rewriting systems

Here we interpret Newman's lemma and the critical pair lemma in the context of $\mathcal{K}$-string rewriting systems.

For a $\mathcal{K}$-string rewriting system (Γ^*, R), denote by $\mathrm{Seq}(\Gamma^*, R)$ the set of rewriting sequences of (Γ^*, R); by $\mathrm{Br}(\Gamma^*, R)$ the set of branchings of (Γ^*, R); and by $\mathrm{Crit}(\Gamma^*, R)$ the set of critical pairs of (Γ^*, R). We denote the length function on $\mathrm{Seq}(\Gamma^*, R)$ by $|\cdot|$. We then have the following result.

Theorem 3.1. *Let (Γ^*, R) be a $\mathcal{K}$-string rewriting system. Then the Kashiwara operators e_i and f_i extend to Seq(Γ^*, R), Br(Γ^*, R), and Crit(Γ^*, R) and commute with the source and target maps of the $\mathcal{K}$-rewriting system. In particular*

i) *for $\mathfrak{s} \in$ Seq(Γ^*, R) and $i \in I$ such that $e_i.\mathfrak{s}$ (resp. $f_i.\mathfrak{s}$) is defined, we have a commutative square*

$$\begin{array}{ccc} w_1 & \xRightarrow{\mathfrak{s}} & w_2 \\ {\scriptstyle i}\uparrow & & \uparrow{\scriptstyle i} \\ e_i.w_1 & \xRightarrow[e_i.\mathfrak{s}]{} & e_i.w_2 \end{array} \qquad \textit{resp.} \qquad \begin{array}{ccc} f_i.w_1 & \xRightarrow{f_i.\mathfrak{s}} & f_i.w_2 \\ {\scriptstyle i}\uparrow & & \uparrow{\scriptstyle i} \\ w_1 & \xRightarrow[\mathfrak{s}]{} & w_2 \end{array}$$

and $|e_i.\mathfrak{s}| = |\mathfrak{s}|$ (resp.$|f_i.\mathfrak{s}| = |\mathfrak{s}|$),

ii) *for a branching $(\alpha, \beta) \in$ Br(Γ^*, R) and $i \in I$ such that $e_i.(\alpha, \beta)$ (resp. $f_i.(\alpha, \beta)$) is defined, we have*

$$\begin{array}{c} (\alpha, \beta) \textit{ is confluent if and only if } e_i.(\alpha, \beta) \textit{ is confluent} \\ \big(\textit{resp. } (\alpha, \beta) \textit{ is confluent if and only if } f_i.(\alpha, \beta) \textit{ is confluent}\big). \end{array}$$

This result shows that the property of termination and of confluence of a $\mathcal{K}$-string rewriting system is independent of the action of the Kashiwara operators. If the $\mathcal{K}$-graph Γ is proper, we can use this result to obtain reduced versions of classical rewriting results in our context as follows. Let (Γ^*, R) be a proper $\mathcal{K}$-string rewriting system and consider an abstract rewriting system $((\Gamma^*)^0, R^0)$ where

$$(\Gamma^*)^0 := \{w \in \Gamma \mid w \text{ a word of highest weight in } \Gamma^*\},$$

and

$$R^0 := \{tuv \overset{t\alpha v}{\Longrightarrow} tu'v \mid tuv \in (\Gamma^*)^0,\ u \overset{\alpha}{\Longrightarrow} u' \in R\}.$$

We have the following consequence of Theorem 3.1.

Corollary 3.2. *Let* (Γ^*, R) *be a proper* $\mathcal{K}$*-string rewriting system. Then* (Γ^*, R) *is terminating respectively (locally) confluent if and only if* $((\Gamma^*)^0, R^0)$ *is terminating respectively (locally) confluent.*

Using this result, we then obtain corresponding $\mathcal{K}$-versions of two classical results in rewriting theory.

Theorem 3.3 (Newman's lemma for $\mathcal{K}$-SRS)**.** *Let* (Γ^*, R) *be a proper* $\mathcal{K}$*-string rewriting system. Then* (Γ^*, R) *is confluent if and only if* $((\Gamma^*)^0, R^0)$ *is terminating and locally confluent.*

To state the Critical Pair Lemma, we remark that the notion of critical pairs descends to the abstract rewriting system $((\Gamma^*)^0, R^0)$. These are the branchings (α, β) with $\alpha, \beta \in R^0$ which are critical in R.

Theorem 3.4 ($\mathcal{K}$-Critical Pair Lemma)**.** *Let* (Γ^*, R) *be a proper* $\mathcal{K}$*-string rewriting system. Then* (Γ^*, R) *is locally confluent if and only if the critical pairs of* $((\Gamma^*)^0, R^0)$ *are confluent.*

3.5 Squier's coherent extension for $\mathcal{K}$-string rewriting systems

Given a convergent string rewriting system X, Squier's theorem [10] asserts that the confluence diagrams of X can be interpreted in terms of a *homotopy basis*, which is a set Ω consisting of confluence diagrams of critical pairs. Note that one may choose different confluence diagrams for Ω. The work of Guiraud and Malbos in [3] gives a deterministic procedure of constructing these base confluence diagrams via normalization strategies, in the case when X is reduced.

In the context of $\mathcal{K}$-string rewriting systems, we have the following interpretation of Squier's coherent completion theorem.

Theorem 3.6 (Squier's theorem for $\mathcal{K}$-string rewriting systems)**.** *Let* (Γ^*, R) *be a convergent* $\mathcal{K}$*-string rewriting system. Then one can choose a coherent completion* (Γ^*, R, Ω) *such that* Ω *admits a* $\mathcal{K}$*-graph structure. Moreover if* Γ *is a proper* $\mathcal{K}$*-graph, then this* Ω *is entirely determined by the confluence diagrams of* $((\Gamma^*)^0, R^0)$*.*

This result, along with Theorems 3.3 and 3.4 reduce the study of confluence of a proper $\mathcal{K}$-string rewriting system (Γ^*, R) to the study of confluence of the abstract rewriting system $((\Gamma^*)^0, R^0)$. In practice, the combinatorics of $\mathcal{K}$-graphs is simplified at words of highest weight, hence the task of studying confluence is easier for $((\Gamma^*)^0, R^0)$.

4 Conclusions

The study of monoids via rewriting theory hinges on two parameters: The first consists of identifying a well-behaved string rewriting system that presents the given monoid; and the second consists of using the combinatorics of the rewriting system and the corresponding monoid to obtain computational results, as for instance expliciting Squier's coherent completion.

The notion of $\mathcal{K}$-string rewriting systems provides a framework for studying $\mathcal{K}$-congruences via adapted string rewriting systems. Firstly, if the $\mathcal{K}$-graph is proper, we obtain versions of Newman's lemma and critical pair lemma in this context, which reduce the verification of properties of termination and confluence of the given $\mathcal{K}$-SRS. Secondly, given a $\mathcal{K}$-convergent string rewriting system, the expliciting of Squier's coherent extension is reduced to computations with words of highest weight.

In [3], Guiraud and Malbos construct a cofibrant replacement for a monoid presented by a convergent presentation. The fact that a $\mathcal{K}$-graph structure and $\mathcal{K}$-string rewriting systems

interact well on free monoids, especially manifested in Squier's coherence theorem, suggests that this behaviour extends to higher dimensions in the context of [3].

References

[1] Alan J Cain, Robert D Gray, and António Malheiro. Crystal monoids & crystal bases: rewriting systems and biautomatic structures for plactic monoids of types an, bn, cn, dn, and g2. *Journal of Combinatorial Theory, Series A*, 162:406–466, 2019.

[2] Mr William Fulton and William Fulton. *Young tableaux: with applications to representation theory and geometry*. Number 35. Cambridge University Press, 1997.

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

[4] Nohra Hage and Philippe Malbos. Knuths coherent presentations of plactic monoids of type a. *Algebras and Representation Theory*, 20(5):1259–1288, 2017.

[5] Masaki Kashiwara. Crystalizing theq-analogue of universal enveloping algebras. *Communications in Mathematical Physics*, 133(2):249–260, 1990.

[6] Jan Willem Klop and JW Klop. *Term rewriting systems*. Centrum voor Wiskunde en Informatica, 1990.

[7] Uran Meha. C-trees and a coherent presentation for the plactic monoid of type c. *arXiv preprint arXiv:2006.03456*, 2020.

[8] Maxwell Herman Alexander Newman. On theories with a combinatorial definition of" equivalence". *Annals of mathematics*, pages 223–243, 1942.

[9] Craige Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of Mathematics*, 13:179–191, 1961.

[10] Craig C Squier, Friedrich Otto, and Yuji Kobayashi. A finiteness condition for rewriting systems. *Theoretical Computer Science*, 131(2):271–294, 1994.

A Example: Plactic monoid of type A

We give here a concrete example of a $\mathcal{K}$-string rewriting system.

Consider the $\mathcal{K}$-graph

$$A_n \ : \ 1 \xrightarrow{1} 2 \xrightarrow{2} 3 \xrightarrow{3} \cdots \xrightarrow{n-2} n-1 \xrightarrow{n-1} n, \tag{3}$$

and set

$$\mathsf{Col}(A_n)_1 := \{w = x_1 x_2 \cdots x_k \mid x_1 < x_2 < \cdots < x_k,\ x_i \in A_n,\ k \leq n\}.$$

Remark A.1. *The $\mathcal{K}$-graph in (3) is called the crystal base of type A_n.*

Such words whose letters are increasing, are called *column words* in A_n^*. The set $\mathsf{Col}(A_n)_1$ satisfies the conditions **(P1)** and **(P2)** hence is a $\mathcal{K}$-graph itself. Define an order $\preceq$ on $\mathsf{Col}(A_n)_1$ by setting $w \preceq w'$ for two columns $w = x_1 \cdots x_k$ and $w' = y_1 \cdots y_l$ if

i) $k \geq l$,

ii) $x_i \leq y_i$ for $i = 1, 2, \ldots, l$.

Schensted's insertion algorithm, as first introduced in [9], and later adapted to a column approach, see [2], describes a procedure of inserting a letter $x \in A_n$ into a column $c_1 \in \mathsf{Col}(A_n)_1$ as follows.

Schensted's algorithm of inserting a letter into a column (SA):

Input: a column $c = x_1 \cdots x_k$; a letter $x \in A_n$;

if $x > x_k$:

set $c' = x_1 x_2 \cdots x_k x$

return: c'

if $x_l \geq x > x_{l-1}$ for some $l \leq k$:

set $c' = x_1 \cdots x_{l-1} x x_{l+1} \cdots x_k$, and $x' = x_l$

return: $x'c'$

The insertion of a letter x into a column c, denoted $(c \leftarrow x)$, outputs either one column, or two columns, with $c = x'$ being the other column. This notion can be extended to an insertion of a letter into a product of two columns as follows

$$(c_1 c_2 \leftarrow x) = \begin{cases} (c_1 (c_2 x)) & \text{if } (c_2 \leftarrow x) \text{ is first case in SA,} \\ (c_1 \leftarrow x_l) c_2' & \text{if } (c_2 \leftarrow x) \text{ is second case in SA.} \end{cases}$$

In [1] Cain, Gray, and Malheiro show that the map $[\ ,\ ] : \mathsf{Col}(A_n)_1^{\times 2} \longrightarrow \mathsf{Col}(A_n)_1^{\times 2}$ defined for $c_1, c_2 \in \mathsf{Col}(A_n)_1$ with $c_2 = x_1 x_2 \cdots x_k$ by setting

$$[c_1, c_2] = (((c_1 \leftarrow x_1) \leftarrow x_2) \leftarrow \cdots) \leftarrow x_k,$$

induces a string rewriting system $\mathsf{Col}(A_n) := (\mathsf{Col}(A_n)_1^*, \mathsf{Col}(A_n)_2)$ where $\mathsf{Col}(A_n)_2$ consists of rewriting rules of the form

$$\mathsf{Col}(A_n)_2 := \{c_1 c_2 \Longrightarrow [c_1, c_2] \mid c_1, c_2 \in \mathsf{Col}(A_n)_1,\ c_1 \not\leq c_2\}.$$

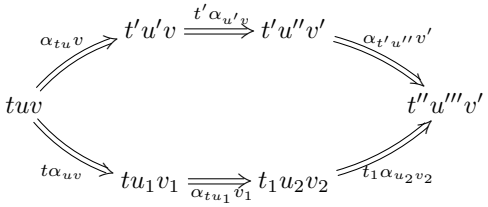
Moreover they prove that this rewriting system is convergent. The monoid presented by $\mathsf{Col}(A_n)$ is called the *plactic monoid of type A*.

We then have the following result.

Theorem A.2 ([1])**.** *The string rewriting system* $\mathsf{Col}(A_n)$ *is a finite reduced convergent* $\mathcal{K}$*-string rewriting system.*

We can then apply Theorem 3.6 to $\mathsf{Col}(A_n)$, and use the combinatorics of $\mathsf{Col}(A_n)$ at highest weight to prove the following.

Theorem A.3. *Squier's homotopy bases for the* $\mathcal{K}$*-string rewriting system* $\mathsf{Col}(A_n)$ *consists of confluence diagrams of the form*



We remark that this result has also been proven by Hage and Malbos in [4] using different techniques.

Confluence Competition 2021

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 10th Confluence Competition (CoCo 2021). 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 12 tools (listed in order of registration) participating in 10 categories (listed in order of first appearance in CoCo):

	TRS	CPF-TRS	CTRS	GCR	UNR	UNC	NFP	COM	INF	SRS
CoLL-Saigawa	✓									✓
CSI	✓	✓			✓	✓	✓			✓
FORT-h				✓	✓	✓	✓	✓		
FORTify	✓			✓	✓	✓		✓		
CO3			✓						✓	
CoLL								✓		
infChecker									✓	
CONFident	✓		✓							✓
NaTT									✓	
ACP	✓	✓	✓			✓		✓		✓
AGCP				✓						
CeTA		✓								

The winning (for combined YES/NO answers) tools[1] of CoCo 2020 participated as demonstration tools, to provide a benchmark to measure progress. The live run of CoCo 2021 on StarExec [2] can be viewed at http://cocograph.uibk.ac.at/2021.html. Further information about CoCo 2021, including a description of the categories and detailed results, can be obtained from

http://project-coco.uibk.ac.at/2021/

References

[1] A. Middeldorp, J. Nagele, and K. Shintani. CoCo 2019: Report on the Eighth Confluence Competition. *International Journal on Software Tools for Technology Transfer*, 2021. doi: 10.1007/s10009-021-00620-4.

[2] 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/2020/results.php.

CoLL-Saigawa 1.6: 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 v1.6 is a joint confluence tool of CoLL v1.5 [9] and Saigawa v1.9 [2], and there are no major changes from the last release (version 1.5). 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 seven confluence criteria: The criteria based on critical pair systems [3, Theorem 3] and on extended critical pairs [5, Theorem 2], rule labeling [11], Church-Rosser modulo AC [4], parallel closedness based on parallel critical pairs [12], simultaneous closedness [7], parallel-upside closedness [8], 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 *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.

CoCo 2021 Participant: CSI 1.2.5

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 1.2.5 contains an implementation of the *outside-closed* criterion for confluence [6] and extends the upside-parallel-closed criterion to include the relaxed condition for root overlaps [5].

CSI participates in the following CoCo 2021 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

[6] M. Oyamaguchi and Y. Ohta. On the Open Problems Concerning Church–Rosser of Left-Linear Term Rewrite Systems. *IEICE Transactions on Information and Systems*, E87-D(1), pages 290—298, 2004.

CoCo 2021 Participant: FORT-h 1.1*

Fabian Mitterwallner, Jamie Hochrainer, and Aart Middeldorp

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

The first-order theory of rewriting is a decidable theory for finite left-linear right-ground rewrite systems. The decision procedure goes back to Dauchet and Tison [1]. FORT-h 1.1 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. More importantly, it can produce certificates for the YES/NO answers. These certificates can then be verified by FORTify, an independent Haskell program that is code-generated from the formalization of the decision procedure in the proof assistent Isabelle/HOL.

A command-line version of FORT-h 1.1 can be downloaded from

http://fortissimo.uibk.ac.at/fort(ify)/

Compared to last year's version, FORT-h 1.1 contains a number of performance improvements. The main ones are smaller intermediate automata constructions due to an earlier elimination of epsilon transitions, and using smaller signature extensions when checking properties on non-ground terms [5].

FORT-h participates in the following CoCo 2021 categories: COM, GCR, NFP, UNC, and UNR. Together with FORTify [6], it participates in the categories COM, TRS, GCR, UNC, and UNR to produce certified YES/NO answers.

References

[1] M. Dauchet and S. Tison. The Theory of Ground Rewrite Systems is Decidable. In *Proc. 5th IEEE Symposium on Logic in Computer Science*, pages 242–248, 1990. doi: 10.1109/LICS.1990.113750.

[2] F. Mitterwallner, A. Lochmann, A. Middeldorp, and B. Felgenhauer. Certifying Proofs in the First-Order Theory of Rewriting. In *Proc. 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 12652 of *LNCS*, pages 127–144, 2021. doi: 10.1007/978-3-030-72013-1_7.

[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.

[5] A. Lochmann, F. Mitterwallner, and A. Middeldorp. Formalized Signature Extension Results for Confluence, Commutation and Unique Normal Forms. In *Proc. 10th International Workshop on Confluence*, 2021. This volume.

[6] A. Lochmann, F. Mitterwallner, and A. Middeldorp. CoCo 2021 Participant: FORTify 1.1. In *Proc. 10th International Workshop on Confluence*, 2021. This volume.

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

CoCo 2021 Participant: FORTify 1.1*

Alexander Lochmann, Fabian Mitterwallner, and Aart Middeldorp

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

The first-order theory of rewriting is a decidable theory for linear variable-separated rewrite systems. The decision procedure goes back to Dauchet and Tison [1]. In this theory confluence-related properties on ground terms are easily expressible. An extension of the theory to multiple rewrite systems, as well as the decision procedure, has recently been formalized [2,3] in Isabelle/HOL. The code generation facilities of Isabelle then give rise to the certifier FORTify [4] which checks certificate constructed by FORT-h [6].

FORTify takes as input an answer (YES/NO), a formula, a list of TRSs, and a certificate proving that the formula holds (does not hold) for the given TRSs. It then checks the integrity and validity of the certificate. Since the first release the formalization was extended to support properties on arbitrary terms, as described in [5]. This allows FORTify to participate, together with FORT-h, in the following CoCo 2021 categories: COM, TRS, GCR, UNC, and UNR.

A command-line version of the tool can be downloaded from

https://fortissimo.uibk.ac.at/fort(ify)/

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] A. Lochmann and A. Middeldorp. Formalized Proofs of the Infinity and Normal Form Predicates in the First-Order Theory of Rewriting. In *Proc. 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 12079 of *LNCS*, pages 178–194, 2020. doi: 10.1007/978-3-030-72013-1_7.

[3] A. Lochmann, A. Middeldorp, F. Mitterwallner, and B. Felgenhauer. Formalizing the First-Order Theory of Rewriting. In *Proc. 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 250–263, 2021. doi: 10.1145/3437992.3439918.

[4] F. Mitterwallner, A. Lochmann, A. Middeldorp, and B. Felgenhauer. Certifying Proofs in the First-Order Theory of Rewriting. In *Proc. 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 12652 of *LNCS*, pages 127–144, 2021. doi: 10.1007/978-3-030-72013-1_7.

[5] A. Lochmann, F. Mitterwallner, and A. Middeldorp. Formalized Signature Extension Results for Confluence, Commutation and Unique Normal Forms. In *Proc. 10th International Workshop on Confluence*, 2021. This volume.

[6] F. Mitterwallner, J. Hochrainer, and A. Middeldorp. CoCo 2021 Participant: FORT-h 1.1. In *Proc. 10th International Workshop on Confluence*, 2021. This volume.

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

CO3 (Version 2.2)

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 rewrite systems (CTRSs, for short) by using a transformational approach (cf. [5]). The tool first transforms a given weakly-left-linear (WLL, for short) 3-DCTRS into an unconditional term rewrite system (TRS, for short) by using $\mathbb{U}_{conf}$ [2], a variant of the *unraveling* $\mathbb{U}$ [8], 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 $\mathbb{U}_{conf}(\mathcal{R})$ 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 [6, 3] to prove infeasibility of a condition w.r.t. a specified CTRS has been implemented [4]. The approach is applicable to *syntactically deterministic* CTRSs that are operationally terminating and *ultra-right-linear* w.r.t. the *optimized* unraveling.

When *join* and *semi-equational* CTRSs are given as input, the previous version returns MAYBE but the present one accepts them as input. To prove confluence of join CTRSs, we consider them as oriented ones [7, Section 5.3].

Theorem 1. *Let $\mathcal{R}$ be a join CTRS, and $\mathcal{R}'$ be $\{\ell \to r \Leftarrow s_1 \twoheadrightarrow x_1, t_1 \twoheadrightarrow x_1, \ldots, s_k \twoheadrightarrow x_k, t_k \twoheadrightarrow x_k \in \mathcal{R} \mid \ell \to r \Leftarrow s_1 \downarrow t_1, \ldots, s_k \downarrow t_k,\ x_1, \ldots, x_k$ are distinkt fresh variables$\}$. Then, (1) $\to_{\mathcal{R}} = \to_{\mathcal{R}'}$, and (2) $\mathcal{R}$ is confluent if and only if $\mathcal{R}'$ is so.*

To prove confluence of semi-equational CTRSs, we consider them as join (i.e., oriented) ones.

Theorem 2. *Let $\mathcal{R}$ be a semi-equational CTRS, and $\mathcal{R}'$ be $\{\ell \to r \Leftarrow s_1 \downarrow t_1, \ldots, s_k \downarrow t_k \in \mathcal{R} \mid \ell \to r \Leftarrow s_1 \leftrightarrow^* t_1, \ldots, s_k \leftrightarrow^* t_k\}$. Then, all of the following hold: (1) $\to_{\mathcal{R}} \supseteq \to_{\mathcal{R}'}$; if $\mathcal{R}'$ is confluent, then (2) $\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}'}$ and (3) $\mathcal{R}$ is confluent.*

Note that the present version does not disprove confluence of join and semi-equational CTRSs.

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. CO3 (Version 2.1). In *Proc. IWC 2020*, page 67, 2020.

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

[6] 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.

[7] N. Nishida, M. Sakai, and T. Sakabe. Soundness of unravelings for conditional term rewriting systems via ultra-properties related to linearity. *Log. Methods Comput. Sci.*, 8(3):1–49, 2012.

[8] 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 1.6: A Commutation Tool

Kiraku Shintani

JAIST, Japan
s1820017@jaist.ac.jp

CoLL (version 1.6) 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 [4]. 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 ([2, 7])**.** *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: simultaneous closedness [5], parallel closedness [9], parallel upside closedness and outside closedness [6], rule labeling with weight function [10, 1], and Church-Rosser modulo A/C [3]. A detailed description of CoLL can be found in [8].

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

[3] 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.

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

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

[6] 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.

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

[8] 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.

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

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

infChecker at the 2021 Confluence Competition*

Raúl Gutiérrez[1], Salvador Lucas[2], and Miguel Vítores[2]

[1] Universidad Politécnica de Madrid, Madrid, Spain
r.gutierrez@upm.es
[2] VRAIN, Universitat Politècnica de València, Valencia, Spain
slucas@dsic.upv.es
mvitvic@posgrado.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 $\bowtie_{ij} \in \{\to, \to^*, \to^+, \hookrightarrow, \hookrightarrow^*, \hookrightarrow^+, \rhd, \trianglerighteq, \downarrow, \downarrow\!\!\!\!\!\!\downarrow, \leftrightarrow, \leftrightarrow\!\!\!\!\!\!\rightarrow, \leftrightarrow^*, \leftrightarrow\!\!\!\!\!\!\rightarrow^*\}$ 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* $\mathsf{Th}_{\bowtie_{ij}}$ [1, 2].

The tool is available here: http://zenon.dsic.upv.es/infChecker/. It is written in Haskell and implements the *Feasibility Framework* [1], that describes:

- *f-problems*, by a tuple $\tau = (\mathbb{T}, \mathcal{G})$, where $\mathbb{T}$ is a $\mathbb{P}$-indexed theory, $\mathbb{P}$ is a set of predicates and $\mathcal{G}$ is a sequence of $\mathbb{T}$-conditions. We say that τ is feasible if $\mathcal{G}$ is $\mathbb{T}$-feasible; otherwise, it is infeasible.
- *f-processor*, as partial functions from f-problems into set of f-problems. Alternatively, it can return "yes". An f-processor $\mathsf{P}(\tau)$ is sound iff $\mathsf{P}(\tau) =$ "yes" or exists $\tau' \in \mathsf{P}(\tau)$ such that τ' is feasible. An f-processor $\mathsf{P}(\tau)$ is complete iff τ is infeasible whenever $\mathsf{P}(\tau) \neq$ "yes" and for all $\tau' \in \mathsf{P}(\tau)$, τ' is infeasible.

We implement five processors: $\mathsf{P}^{\mathsf{Sat}}$ to prove infeasibility, $\mathsf{P}^{\mathsf{Prov}}$ to prove feasibility, P^{NC} to apply narrowing, $\mathsf{P}^{\mathsf{Spl}}$ to decompose a goal and P^{UR} to simplify the set of rules.

Our feasibility problems accepts different variants of TRS: Term Rewriting Systems, Conditional Term Rewriting Systems, Context-Sensitive Term Rewriting Systems and Conditional Context-Sensitive Term Rewriting Systems.

By using the proper relation (straight arrows for rewriting and curly arrows for context-sensitive rewriting) we can use and combine different forms of rewriting.

References

[1] R. Gutiérrez and S. Lucas. Automatically Proving and Disproving Feasibility Conditions. In N. Peltier and V. Sofronie-Stokkermans, editor, *Proc. of IJCAR'2020*, LNCS 12167:416–435. Springer, 2020.

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

*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.

CONFident at the 2021 Confluence Competition*

Miguel Vítores[2], 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
mvitvic@posgrado.upv.es
slucas@dsic.upv.es

1 Overview

CONFident 1.0 is a tool for checking the *confluence* or *non-confluence* of systems based on rewriting by means of its logical representation. The tool is available here: http://zenon.dsic.upv.es/confident/. It is written in Haskell following a DP-framework structure, i.e., by defining problems and processors:

- problems are tuples $\tau = (\mathbb{T}, \mathcal{G})$, where $\mathbb{T}$ is a $\mathbb{P}$-indexed theory, $\mathbb{P}$ is a set of predicates and $\mathcal{G}$ are logical goals representing conditions to be checked during the analysis: joinability, reachability, feasibility, etc. We can use predicate symbols as $\rightarrow$ and $\rightarrow^*$ to represent different kind of relations between terms defined by a logical theory. We say that τ is confluent if $\mathcal{G}$ is $\mathbb{T}$-confluent; otherwise, it is non-confluent.
- processors are defined as partial functions from problems into set of (hopefully simpler) problems. Our processors are based on transforming problems of confluence into logical problems that can be solved by external tools (infeasibility checkers, theorem provers, model generators...).

We implement these processors using the logical approach presented in [1, 3, 4] and mechanizing them by external tools like MU-TERM [3], infChecker [1], AGES [2], Prover9 and Mace4 [6] and Barcelogic[1].

References

[1] R. Gutiérrez and S. Lucas. Automatically Proving and Disproving Feasibility Conditions. In N. Peltier and V. Sofronie-Stokkermans, editor, *Proc. of IJCAR'2020*, LNCS 12167:416–435. Springer, 2020.

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

[3] R. Gutiérrez and S. Lucas. MU-TERM: Verify Termination Properties Automatically (System Description). In N. Peltier and V. Sofronie-Stokkermans, editor, *Proc. of IJCAR'2020*, LNCS 12167:436–447. Springer, 2020.

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

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

[1] https://barcelogic.com/

NaTT 2.2 in CoCo 2021

Akihisa Yamada

National Institute of Advanced Science and Technology

`NaTT` [4] is a termination prover for plain term rewriting. It is written in OCaml and the source code is available at:

`https://www.trs.cm.is.nagoya-u.ac.jp/NaTT/`

Though it has nothing to do with proving confluence, `NaTT` implements a quick reachability check [3] for computing estimated dependency graphs [1]. To demonstrate the strength (or more precisely, weakness) of this reachability check, this year `NaTT` will participate in the "infeasibility" category of the Confluence Competition. Infeasibility means negated reachability, which can be tested by the aforementioned method. To meet the specification of the category, `NaTT` had to be modified to be able to

- expose the reachability checking function, and
- parse the COPS format for infeasibility.

An interesting task was the latter. To this end, the author incorporated a generic text-to-and-from-XML translator that he developed for another project, in order to translate the COPS format into a newly defined simple XML format for TRSs, which `NaTT` can understand. As a positive side effect, `NaTT` can now directly read the (complex) XML problem format of the Termination Competition [2]. A negative side effect is that the binary `bin/NaTT.exe` of version 2.2 does not read the old WST format anymore, but the script `bin/NaTT.sh` does.

At this point, it turned out that most of the infeasibility problems in COPS database are conditional TRSs. Therefore, the author had further to parse conditional rules. This was easy thanks to the above XML translator. However, as `NaTT` is for plain term rewriting, conditions are simply ignored. Thus it will only answer YES (unreachable) if unreachability could be proved without conditions, and will never answer NO (reachable).

References

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

[2] Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The termination and complexity competition. In Dirk Beyer, Marieke Huisman, Fabrice Kordon, and Bernhard Steffen, editors, *TACAS 2019 (3)*, volume 11429 of *LNCS*, pages 156–166. Springer, 2019.

[3] Christian Sternagel and Akihisa Yamada. Reachability analysis for termination and confluence of rewriting. In Tomás Vojnar and Lijun Zhang, editors, *TACAS 2019, Part I*, volume 11427 of *LNCS*, pages 262–278. Springer, 2019.

[4] Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. Nagoya Termination Tool. In Gilles Dowek, editor, *RTA-TLCA 2014*, volume 8560 of *LNCS*, pages 466–475. Springer, 2014.

ACP: System Description for CoCo 2021

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 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 criterion has been incorporated from the one submitted for CoCo 2020.

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 2021

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 $u\theta_g \overset{*}{\underset{\succsim}{\longleftrightarrow}}_{\mathcal{R}} v\theta_g$ for any its ground instance $u\sigma_g \approx v\sigma_g$, where $x \overset{*}{\underset{\succsim}{\longleftrightarrow}} y$ iff there exists $x = x_0 \leftrightarrow \cdots \leftrightarrow x_n = y$ such that $x \succsim x_i$ or $y \succsim x_i$ for every x_i.

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

No new ground (non-)confluence criterion has been incorporated from the one submitted for CoCo 2020.

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 2021 Participant: CeTA 2.40*

René Thiemann[1]

University of Innsbruck, Austria

The tool CeTA [1] 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 the relevant changes of version 2.40 of CeTA w.r.t. confluence proving. Although there are no new dedicated confluence techniques in CeTA, we like to mention an extended support for termination proofs. This extension has the potential to increase the power of confluence techniques that rely upon termination or relative termination.

In particular there is an extension that consists of a generalization of the weighted path order (WPO) [3, 2], where we now added support for multiset comparisons (cases (2c) and (2d)). Consequently, the generalized version of WPO subsumes the recursive path order.

Definition 1 (Generalized WPO)**.** *Let $\mathcal{A}$ be a weakly monotone algebra over signature Σ, $\succsim$ a precedence, π a status, and let $c : \Sigma \to \{\mathsf{lex}, \mathsf{mul}\}$. Let $\geq_{\mathcal{A}}$ be simple w.r.t. π. The WPO reduction pair $(\succ_{\mathsf{WPO}}, \succsim_{\mathsf{WPO}})$ is defined as follows: $s \succ_{\mathsf{WPO}} t$ iff $s >_{\mathcal{A}} t$, or $s \geq_{\mathcal{A}} t$ and*

1. *$s = f(s_1, \ldots, s_n)$ and $\exists i \in \pi(f).\ s_i \succsim_{\mathsf{WPO}} t$, or*
2. *$s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_m)$, $\forall j \in \pi(g).\ s \succ_{\mathsf{WPO}} t_j$ and*
   - *(a) $f \succ g$,*
   - *(b) $f \succsim g$ and $c(f) = c(g) = \mathsf{lex}$ and $\pi(f)[s_1, \ldots, s_n] \succ^{\mathsf{lex}}_{\mathsf{WPO}} \pi(g)[t_1, \ldots, t_m]$,*
   - *(c) $f \succsim g$ and $c(f) = c(g) = \mathsf{mul}$ and $\pi(f)[s_1, \ldots, s_n] \succ^{\mathsf{mul}}_{\mathsf{WPO}} \pi(g)[t_1, \ldots, t_m]$, or*
   - *(d) $f \succsim g$ and $c(f) \neq c(g)$ and $\pi(f)[s_1, \ldots, s_n] \neq [\,]$ and $\pi(g)[t_1, \ldots, t_m] = [\,]$.*

The relation $s \succsim_{\mathsf{WPO}} t$ is defined in a similar way and we refer to theory `Orderings/WPO_MS.thy` within IsaFoR for the full formal definition.

We would like to welcome all confluence tool developers to experiment whether our new extension is indeed helpful for confluence proving, and are looking forward to certify these new kinds of proofs via CeTA.[1]

References

[1] 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. 2009.

[2] 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*, volume 167 of *LIPIcs*, pages 4:1–4:20, 2020.

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

*This work was supported by the Austrian Science Fund (FWF) project Y757.

[1] The CPF-format has not yet been extended to cover the extension of WPO to multisets. We invite all tool authors to get in contact with us to fix the precise format for WPO with multiset comparisons.

Author Index