IWC 2018

7th International Workshop on Confluence

Proceedings

Editors: Bertram Felgenhauer & Jakob Grue Simonsen

July 7th, 2018, Oxford, United Kingdom

Preface

This report contains the proceedings of the 7th International Workshop on Confluence (IWC 2018) in Oxford, United Kingdom on July 7th, 2018. The workshop is part of the Federated Logic Conference (FLoC 2018), associated with the 3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018). Previous IWC workshops were held in Nagoya (2012), Eindhoven (2013), Vienna (2014), Berlin (2015), Obergurgl (2016), and Oxford (2017).

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

IWC 2018 received 11 submissions. Most submissions were reviewed by 3 program committee members. After deliberations, the program committee decided to accept 8 submissions, which are contained in this report. Apart from these contributed talks, the workshop has one invited talk by Maja Kirkeby and Henning Christiansen, about Confluence in Constraint Handling Rules. Their extended abstract is included in this report.

Several people contributed to IWC 2018 preparations. We are grateful to the members of the program committee for their work. We also thank the members of the FLoC and FSCD organizing committees for hosting IWC 2018 in Oxford.

May 31, 2018
Copenhagen / Innsbruck

Jakob Grue Simonsen
Bertram Felgenhauer

Update (June 18, 2018): This version includes the system descriptions of the 7th Confluence Competition (CoCo 2018). Note, however, that CoCo will run during the FSCD conference this year. For further information please refer to the CoCo website at

http://coco.nue.ie.niigata-u.ac.jp/2018/

Program Committee Chairs

Bertram Felgenhauer	University of Innsbruck
Jakob Simonsen	University of Copenhagen

Program Committee

Bertram Felgenhauer	University of Innsbruck
Jeroen Ketema	TNO-ESI
Kentaro Kikuchi	Tohoku University
Samuel Mimram	École Polytechnique
Julian Nagele	Queen Mary University of London
Jakob Simonsen	University of Copenhagen

Table of Contents

CoCo System Descriptions

Confluence in Constraint Handling Rules: A retrospective overview

Henning Christiansen and Maja H. Kirkeby

Computer Science, Roskilde University, Denmark
henning@ruc.dk and majaht@ruc.dk

1 Introduction

Constraint Handling Rules, CHR, is a nondeterministic programming language whose programs consists of rewrite rules over program states, and being able to show confluence may be an important part of a program correctness proof. Confluence of has been studied from the first introduction of CHR [8, 9]. The first essential results on proving confluence in CHR [1–3], developed during the 1990s, were formulated with respect to a logic based semantics. This choice gave elegant proofs for terminating systems based on the subsumption principle. More recent work extends the previous methods for proving confluence to include invariants and confluence modulo equivalence. Furthermore, these results were developed for a more realistic semantics that reflects the de-facto standard implementations of CHR upon Prolog, including a correct treatment of Prolog's non-logical devices (e.g., var/1, nonvar/1, is/2) and runtime errors. In the following we give an overview of confluent results for Constraint Handling Rules, from early, fundamental results to recent extensions including state invariants and confluence modulo equivalence.

2 Preliminaries

We rephrase the following standard definitions and properties. A *transition system* $D = \langle S, \mapsto \rangle$ consists of a set of *states* S; a *transition* is an element of $\mapsto : S \to S$, written $s_1 \mapsto s_2$ or, alternatively, $s_2 \mapsfrom s_1$, and $\mapsto^*$ is the reflexive transitive closure of $\mapsto$. An *object corner* is a structure of the form $s_1 \mapsfrom s \mapsto s_2$ in which the indicated relationships hold. A system is *confluent* whenever, for all s, s_1, s_2 with $s_1 \mapsfrom^* a \mapsto^* s_2$, that s_1, s_2 are joinable, i.e. there exists a state t such that $s_1 \mapsto^* t \mapsfrom^* s_2$; and it is *locally* confluent whenever any object corner is joinable. Newman's lemma: A terminating system is confluent if and only if it is locally confluent.

Proving different types of systems confluent has been facilitated by constructions of sets of critical pairs – or *critical corners* as we refer to, including the common ancestor state. Typically, these critical corners are selected object corners of the same system (but we will relax this later) and are accompanied by a notion of *subsumption* (that may vary depending on the type of system), i.e., a given critical corner subsumes a set object corners.

The set of critical corners should satisfy the following properties[1] - whenever a critical corner is joinable, any object corner that it covers is joinable, - an object corner not covered by a critical corner is known to be joinable in some way or another (and thus referred to as trivial corners). Thus, a proof of confluence for a terminating system may a matter of 1) describe a set of critical corner, 2) show each of them joinable; we refer to this proof strategy as the

[1] E.g., as a consequence of a property called the Critical pair lemma, that we no not exactly rephrase here.

subsumbtion principle. Ideally a set of critical corners is finite, but this may not always be the case.

3 Constraint Handling Rules

Constraint Handling Rules [8, 10, 11], CHR, is a nondeterministic programming language based on rewriting rules. It was originally designed for constraint solving but has become important as a general-purpose language for representing knowledge and expressing algorithms in a high-level fashion; today it is applied in many areas, for instance analysis of types, multi-agent systems, scheduling, and abductive reasoning; see, e.g., [12] for an overview. While most common CHR implementations are deterministic rather than nondeterministic, it is still useful to consider CHR programs as nondeterministic as it allows the programmer to disregard the execution model of the specific implementation.

CHR relates to the logic programming tradition; constraints are first-order atoms and the language has a declarative semantics [11] based on a logical reading of the rules. CHR programs consist of (a finite set of) guarded rewrite rules over multi-sets of constraints, called *constraint stores*.

Example 1 ([4, 5]). The following CHR program, consisting of a single rule (without guard), collects a number of separate items into a set represented as a list of items.

```
set(L), item(A) <=> set([A|L]).
```

This rule will apply repeatedly, replacing constraints matched by the left hand side by those indicated to the right, there exists a substitution from the rule constraints to the constraint store constraints. The query

```
?- item(a), item(b), set([]).
```

may lead to two different final states, {`set([a,b])`} and {`set([b,a])`}, both representing the same set.

There are three rule types in CHR; the one in the example above is a *simplification* rule that replaces constraints by new constraints; another rule type called *propagation* rule adds new constraints (such a rule is written using a different arrow '`==>`'); and the last type, called a *simpagation*, is a generalization of these two. CHR applies a bookkeeping mechanism to avoid trivial looping that otherwise arises with propagation rules. This is easily incorporated into a formal semantics, but for simplicity we have decided to ignore this. For an in-depth introduction of all rule types, see e.g. [11].

There are two sorts of constraints user-defined that are those appearing in the head of the rules, and built-in constraints. The exact set of available built-in constraints and the modeling of their meaning varies with the CHR semantics of interest. Several CHR semantics have been proposed [1–7, 11]. The semantics used for confluence considerations has traditionally only allowed logical built-ins [1–3, 6, 7, 11], but a recent semantics [4, 5] reflects the de-facto standard implementations of CHR upon Prolog, including a correct treatment of Prolog's non-logical devices (e.g., var/1, nonvar/1, is/2) and runtime errors.

The shape of a CHR state varies with the semantics: in a simple[2] logic-based semantics, a state is of the form $\langle S, B\rangle$ where S is a constraint store and B is a built-in store containing a conjunction of built-ins where each built-in has a logical meaning; and in a Prolog-based

[2]When states were introduced originally [1–3], they contained several extra state components, but these extra components were shown redundant [4, 5].

semantics, a state consists of a constraint store S and each built-in has an operational meaning producing a special substitution that is applied to S. For instance t_1>t_2 evaluates to an empty substitution if t_1 and t_2 are ground arithmetic expressions with values v_1 and v_2, and $v_1 > v_2$ holds, it evaluates to a *failure* substitution if instead $v_1 \not> v_2$, and to an *error* substitution, otherwise. Applying an *error* (a *failure*) substitution to a state change the state to a special state, namely an *error*-state (a *failure*-state); if such a state is reached no further transformations are allowed; therefore, the Prolog-based semantics is sensitive to the execution order of built-ins.

A built-in constraint may occur in the constraint store either introduced in the start query or by a rule body, and in rule-guards; user-defined constraints may not occur in the guards and built-in constraints may not occur on the rules left-hand sides. In the logic based semantics, built-ins in the constraint store are transferred to the built-in store, and in the Prolog based semantics they are evaluated by their operational meaning and the state is updated accordingly. For instance, considering the logic based semantics $\langle\{a{=}X, p(X)\}, \mathit{true}\rangle$ is updated to $\langle\{p(X)\}, X{=}a\rangle$ and considering instead the Prolog based semantics $\langle\{a{=}X, p(X)\}\rangle$ is transformed to $\langle\{p(a)\}\rangle$, see, e.g., [11] and respectively [5] for definitions.

A guard is a sequence of built-in constraints[3]; in the above example there are no guards, but the following example includes rule-guards on the right-hand side, e.g. (`X>0 |`). For the logic based semantics a rule may be applied if the built-in store implies the guard; and in the Prolog based semantics it may be applied if a rule-guard evaluates to neither failure nor an error, and it does not instantiate existing constraint store variables.

Example 2. Consider the following CHR program with four rules, r_1–r_4.

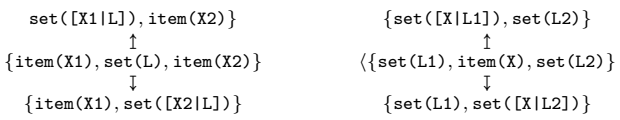
r_1:	`p(X) <=> q(X)`	r_3:	`q(X) <=> X>0 | r(X)`
r_2:	`p(X) <=> r(X)`	r_4:	`r(X) <=> X≤0 | q(X)`

In both semantics r_3 applies to `q`(n) constraints when n is a positive number, e.g., `q(1)`. Under the Prolog based semantics the rule cannot apply to the state $\langle\{$`q(X)`$\}\rangle$ since the variable `X` causes the guard `X>0` to result in an error. Under the logic based semantics the r_3 may transform a state $\langle\{$`q(X)`$\}, $`X>2`$\rangle$ to $\langle\{$`r(X)`$\}, $`X>2`$\rangle$ because `X>0` is a logical consequence of `X>2`, whereas r_3, for instance, does not transform the state $\langle\{$`q(X)`$\}, \mathit{true}\rangle$.

4 Confluence in CHR

The results on confluence for CHR are similar to those for term rewriting systems; critical corners appear when two instances of rules can apply to overlapping constraints in the constraint store, see, e.g., following example.

Example 3 (Ex. 1 continued). The `set`-program of Example 1 is not confluent under neither semantics[4], as both critical corners



are not joinable. Note that the built-in store would be *true* under the logic based semantics.

[3] In the logic based semantics it is seen as a conjunction. In the Prolog based semantics they are evaluated from left to right, each influencing the whole state including the subsequent series of the built-ins; if the guard evaluates to failure or error these updates are discarded, see [5] for details.

[4] The built-in store under the logic based semantics is *true* for all indicated states.

Example 4 (Ex. 2 continued). The program of Example 2 is not confluent under neither semantics[4], as its single critical corner $\{\texttt{q(X)}\} \leftarrow\!\shortmid \{\texttt{p(X)}\} \mapsto \{\texttt{r(X)}\}$ is not joinable.

When a critical corner is formed by rules with guards, their satisfaction is incorpotated into the common ancestor state, which works nicely under the logic based semantics when only logical built-ins are assumed. Here subsumption of an object corner by a critical corner is defined by applying substitution and adding more constraints; the inherent monotonicity ensures joinabilty of any object corner subsumed by a critical corner.

However, the subsumption principle as explained so far as well as the use of the logic based semantics cannot handle not-logical built-in predicates that are available – and extensively used in practice – in standard implementations of CHR. Consider, for example, the CHR rule `p(X) <=> var(X) | q(X)`, whose guard is consists of Prolog's test for whtter its argument is an uninstantiated variable. While the rule may apply to a state containing `p(X)`, it does not apply to a more specific state containing `p(1)`. The other way round, if the guard is instead `nonvar(X)`, the rule may apply to a lot of subsumed instanced with `p(1)`, `p(2)`, ..., but this cannot be "verified" by investigating a most general state including `p(X)` to which the `nonvar` version of the rule does not apply.

In order to restore a subsumption principle aiming at finite proofs, we have taken the consequence in our own work to describe critical corners in a different system with higher expressibility than the object system. In the informal example considered above we can formally characterize — as a meta-level state – expressions like"`p(`x`)` where x is a variable", as well as "... x is a constant", and perform meta-level transitions. As it is shown below, the use of such a meta-level representation is also a powerful tool when confluence under invariant and/or modulo equivalence, which otherwise has been problematic, even for the logical subset of CHR under a logic-based semantics.

5 Invariants and modulo equivalence

It can be argued that invariants and confluence modulo equivalence are important from a practical point of view. In this section we give definitions and examples and later we consider how to prove the properties. Most programs are developed with a particular set of initial queries in mind, which reduces the set of reachable states. In 2007, Duck et al. [7] suggested to take such an induced invariant into account and, thus, make a much larger class of programs confluent.

Definition 1. A set I is a state *invariant* for a relation $\mapsto$ if $x \in I \wedge x \mapsto y$ implies $y \in I$.

Such an invariant may be *induced* by a set of reachable states from a set of (initial) states Q, i.e. $I = \{s' \mid s \in Q \wedge s \mapsto^* s'\}$.

Example 5 (Ex. 1 continued). The *set*-program reflects a tacitly assumed state invariant: only one `set`-constraint is allowed. If we open up for a query such as

```
?- item(a), item(b), set([]), set([c]).
```

we obtain a collection of different answers, representing different ways of partitioning {`a`, `b`, `c`} into two sets. However, this may not be intended, and the relevant invariant $I_{\texttt{set}}$ must specify that a state must include at most one `set`/1 constraint and a series of `item`/1 constraints.

Definition 2. A relation $\mapsto$ is *observable confluent* (under invariant I) if and only if $\forall x, y, y' \in I\colon y' \leftarrow\!\shortmid^* x \mapsto^* y' \Rightarrow \exists z \in I\colon y' \mapsto^* z \leftarrow\!\shortmid^* y'$. We may write I-*observable confluent* meaning observable confluent (under invariant I).

Example 6 (Ex. 2 continued). Consider the program of Example 2 together with an invariant $I_{>0}$ induced by initial states with a single atom `p(X)` where n is a positive number (not a variable). The program is $I_{>0}$-observable confluent, since each forked state $\mathtt{q}(n) \leftarrow\!\shortmid\ \mathtt{p}(n) \mapsto \mathtt{r}(n)$ is joinable by rule r_3: $\mathtt{r}(n) \mapsto \mathtt{q}(n)$.

Confluence modulo equivalence is a generalization where forked states must reach equivalent states, rather than a common state. For instance, a program may produce redundant data structures such as representing sets as lists, and the equivalence states that the order of the elements does not matter. Confluence modulo equivalence was first considered for CHR in 2014 by Christiansen and Kirkeby [4].

Definition 3. A relation $\mapsto$ is *confluent modulo an equivalence* $\approx$ if and only if $\forall x, y, x', y' : y' \leftarrow\!\shortmid^* x' \approx x \mapsto^* y' \Rightarrow \exists z, z' : y' \mapsto^* z' \approx z \leftarrow\!\shortmid^* y'$.

Example 7 (Ex. 1, 5 continued). The *set*-program is supposed to produce one set representation, and we introduce a state equivalence $\approx_{\mathtt{set}}$ reflecting the redundant data structures. Two states are equivalent if they have the same `item`-constraints and their respective `set`-constraint `set(`L_1`)` and `set(`L_2`)` are such that L_1 and L_2 are permutations of each other.

We generalize confluence modulo equivalence and observable confluence as follows.

Definition 4. A relation $\mapsto$ is *I-observable confluent modulo an equivalence* $\approx$ if and only if $\forall x, y, x', y' \in I : y' \leftarrow\!\shortmid^* x' \approx x \mapsto^* y' \Rightarrow \exists z, z' \in I : y' \mapsto^* z' \approx z \leftarrow\!\shortmid^* y'$.

Both observable confluence, confluence modulo equivalence and classic confluence are special cases of this definition.

Huet [14] provided a pair of local properties for showing terminating programs confluent modulo equivalence; we extend these with an invariant as follows.

Definition 5. A rewriting system $\mapsto$ is *locally I-observable confluent modulo* $\approx$ if and only if it has the following α- and β-properties.

$$\begin{array}{lllll} \alpha: & \forall x, y, y' \in I: & y' \leftarrow\!\shortmid x \mapsto y' & \Rightarrow & \exists z, z' \in I: \quad y' \mapsto^* z' \approx z \leftarrow\!\shortmid^* y' \\ \beta: & \forall x, y, y' \in I: & y' \approx x \mapsto y' & \Rightarrow & \exists z, z' \in I: \quad y' \mapsto^* z' \approx z \leftarrow\!\shortmid^* y' \end{array}$$

We refer to structures of the form $y' \leftarrow\!\shortmid x \mapsto y'$ as α-corners and those of the form $y' \approx x \mapsto y'$ as β-corners.

Theorem 1 (obs. confl. mod. equivalence)**.** *A terminating relation $\mapsto$ is I-observable confluent modulo $\approx$ if and only if it is locally I-observable confluent modulo $\approx$.*

In the special cases where equivalence is '=', the β-property trivially holds and when, furthermore, the invariant is unrestrictive the α-property reduces to local confluence.

Both the invariant and the equivalence relation may be tailored for the individual program. By nature, they are meta level properties that in general cannot be expressed in its own system: the state itself is implicit and properties such as groundness (or certain arguments bound to be variables) cannot be expressed in a logic-based semantics for CHR.

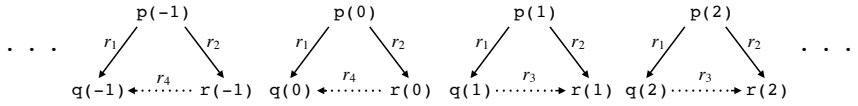
6 Proving observable confluence

As mentioned, observable confluence was introduced by Duck et al. [7]. They suggested methods of proving this property for logic CHR programs using a logic based semantics and as a direct continuation of the logic subsumption principle.

Firstly, they construct the set of critical corners based from the program rules as explained in Section 4. Typically, the states in these corners do not satisfy the invariant (a rule typically includes variables, contradicting groundness), and the next step is to characterize a set of "minimal extensions" of each critical corner such that 1) the states of such an extension satisfies the invariant, 2) the set of all such extensions subsumes (by substitution and adding constraints) all non-trivial object corners. Proving observable confluence amounts to show joinability af such extension, using the standard transition relation for CHR.

There are two problems in this approach, first of all there is no formal representation of the invariant that allows to take it into account when reasoning formally about joinability, and – more importantly – as also noticed by Duck et al. [7], quite often there is an infinite number of such extensions. This happens even for an intuitively simple invariant such as requiring ground states. We can demonstrate this phenomenona for the program of Example 2.

Example 8. Consider the program of Example 2; it is not confluent as its single critical corner `q(X)` $\leftarrow\!\!\shortmid$ `p(X)` $\shortmid\!\!\rightarrow$ `r(X)` is not joinable (the built-in store is always *true* and thus omitted). However, adding an invariant "reachable from an initial state `p(`n`)` where n is an integer" makes it confluent. We indicate the smallest set of corners found by minimal extensions of the critical corner; the dotted transitions prove each of them joinable:

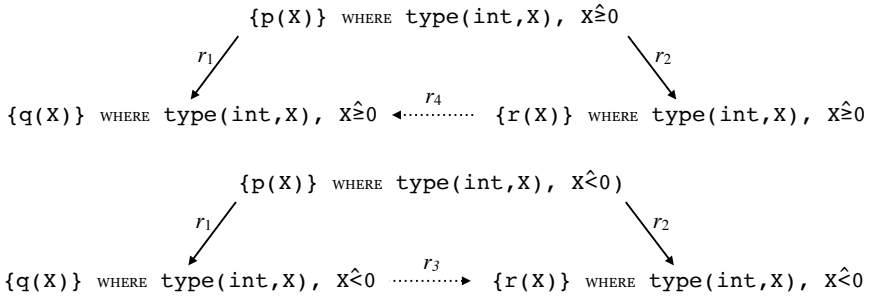


This set is exactly the set of all non-trivial object level corners. These corners and their proofs of joinability obviously fall in two groups of similar shapes, but there is no way to construct a finite set (of, say, one or two elements) of critical corners in CHR, that covers all object corner.

To avoid this problem, Christiansen and Kirkeby [15] suggests to describe critical corners in a more powerful meta-language rather than using CHR itself, inspired by earlier work on meta-programming in logic programming. Each term of CHR is here named by a ground term, specifically, variables named by ground constants. A variable in such a term is thus a meta-variable, which may be covered by a meta-level constraint. For example, the meta-level term `p(`x`)` WHERE `type(const,`X`)` subsumes all object level (=CHR) atoms whose predicate is `p`/1 and whose argument is a constant, i.e., `p(a)`, `p(b)`, ..., `p(1)`, ... The authors define a notion of abstract similation making precise what it means for meta-level transitions and corners to subsume[5] sets of object level transitions and corners. Built-in predicates are reflected at the meta-level, such that, say $n\widehat{<}0$ restricts n to names of terms t that satisfy the object level condition $t < 0$, i.e., n is limited names of number constrants less that zero.

Example 9. (Continuing Ex. 2, 8) The invariant is formalized at the meta-level as states of the form $\langle\{$*pred*`(`n`)`$\}, \mathit{true}\rangle$ WHERE `type(int,`n`)` where *pred* is one of `p`, `q` and `r`. Below is shown the two joinable critical meta-level corners that can be shown to subsume all non-trivial object level corners.

[5] [15] use the terminology of a meta-level term *covering* object level notions.



This example illustrates an additional technique called splitting: First a meta-level corner is produced in the classical way, considering how rules can overlap; this yields a common meta-level ancestor state `p(`x`)` WHERE `type(int,`x`)` and the other states as above containing `q(`x`)`, resp. `r(`x`)`. This meta-level corner is in itself not joinable as no single rule can apply, but turning it into two meta-level corners, each joinable and together subsuming the same set of object-level corners, proves observable confluence.

7 Proving Confluence modulo equivalence

Confluence modulo equivalence has been studied since the first half of the 20th century in a variety of contexts; see, e.g., [5, 14] for an overview. It was introduced and motivated for CHR by [4], also arguing that invariants are important for specifying meaningful equivalences. An in-depth theoretical analysis, including the use of the ground representation, is given by [5] in relation the Prolog-related semantics mentioned above.

To show confluence modulo equivalence for terminating CHR programs, two types of critical corners must be constructed: *critical α-corners* which are the standard critical corners constructed by rule overlap as above, and *critical β-corners* of the form $y' \approx x \mapsto y'$, cf. Definition 5. As before the critical β-corners must subsume all non-trivial object-level β-corners. The meta-level language described above is also suitable for describing state equivalences [15].

Example 10 (Ex. 1 continued). The `set`-program of Example 1 is observable confluent modulo $\approx_{\texttt{set}}$ (Ex. 7) under invariant $I_{\texttt{set}}$ (Ex. 5) since the critical α-corner with two `set`-constraints does not subsume $I_{\texttt{set}}$ corners and both the other critical α-corner and the critical β-corner are joinable modulo $\approx_{\texttt{set}}$, see below. The meta-level constraint `its`/1 constrains its argument to a set of `item`-constraints and `perm`/2 constrains the arguments to a pair of permuted lists.
α-corner:

$$\begin{array}{ccc} \{\texttt{set([X1|L])},\texttt{item(X2)}\} \uplus C \text{ WHERE } \texttt{its}(C) & \mapsto & \{\texttt{set([X2,X1|L])}\} \uplus C \text{ WHERE } \texttt{its}(C) \\ \updownarrow & & \\ \{\texttt{item(X1)},\texttt{set(L)},\texttt{item(X2)}\} \uplus C \text{ WHERE } \texttt{its}(C) & & \wr \\ \downarrow & & \\ \{\texttt{item(X1)},\texttt{set([X2|L])}\} \uplus C \text{ WHERE } \texttt{its}(C) & \mapsto & \{\texttt{set([X1,X2|L])}\} \uplus C \text{ WHERE } \texttt{its}(C) \end{array}$$

β-corner:

$$\begin{array}{ccc} \{\texttt{set([L2])},\texttt{item(X)}\} \uplus C \text{ WHERE } \texttt{perm(L1,L2)} \wedge \texttt{its}(C) & \mapsto & \{\texttt{set([X|L2])}\} \uplus C \text{ WHERE } \texttt{perm(L1,L2)} \wedge \texttt{its}(C) \\ \wr & & \\ \{\texttt{set([L1])},\texttt{item(X)}\} \uplus C \text{ WHERE } \texttt{perm(L1,L2)} \wedge \texttt{its}(C) & \curvearrowright & \\ \downarrow & & \\ \{\texttt{set([X|L1])}\} \uplus C \text{ WHERE } \texttt{perm(L1,L2)} \uplus C \wedge \texttt{its}(C) & & \end{array}$$

A recent paper [13] attempts to handle (observable) confluence modulo equivalence within the logic-based semantics, along the lines of Duck et al [7]. However, this implies the mentioned

problems of infinitely many proof cases, which seems to be inherent in relying on pure logic-based subsumption without having the enhanced expressibility provided by a suitable meta-level representation.

8 Future work

We have given an overview of classic and recent results for confluence in CHR. The classic results provide a theoretical foundation and the recent results on observable confluence and modulo equivalence points towards more practical applications of these notions in a programming context.

There exist methods for automatic check of confluence for CHR [16] in a strictly logical setting, and in our own work we are developing similar methods for automatic or semi-automatic proofs of observable confluence modulo equivalence. Naturally, invariants and state equivalences may involve undecidability.

References

[1] S. Abdennadher. Operational semantics and confluence of constraint propagation rules. In *CP*, pages 252–266, 1997.

[2] S. Abdennadher, T. W. Frühwirth, and H. Meuss. On confluence of Constraint Handling Rules. In *CP*, volume 1118 of *LNCS*, pages 1–15. Springer, 1996.

[3] S. Abdennadher, T. W. Frühwirth, and H. Meuss. Confluence and semantics of constraint simplification rules. *Constraints*, 4(2):133–165, 1999.

[4] H. Christiansen and M. H. Kirkeby. Confluence modulo equivalence in constraint handling rules. In *LOPSTR 2014*, volume 8981 of *LNCS*, pages 41–58, 2015.

[5] H. Christiansen and M. H. Kirkeby. On proving confluence modulo equivalence for constraint handling rules. *Formal Aspects of Computing*, 29(1):57–95, 2017.

[6] G. J. Duck, P. J. Stuckey, M. J. G. de la Banda, and C. Holzbaur. The refined operational semantics of Constraint Handling Rules. In *ICLP 2004*, volume 3132 of *LNCS*, pages 90–104. Springer, 2004.

[7] G. J. Duck, P. J. Stuckey, and M. Sulzmann. Observable confluence for Constraint Handling Rules. In *ICLP*, volume 4670 of *LNCS*, pages 224–239. Springer, 2007.

[8] T. W. Frühwirth. User-defined constraint handling. In *ICLP*, pages 837–838. MIT Press, 1993.

[9] T. W. Frühwirth. Constraint handling rules. In *Constraint Programming: Basics and Trends, Châtillon Spring School, Châtillon-sur-Seine, France, May 16 - 20, 1994, Selected Papers*, volume 910 of *LNCS*, pages 90–107. Springer, 1994.

[10] T. W. Frühwirth. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1-3):95–138, 1998.

[11] T. W. Frühwirth. *Constraint Handling Rules*. Cambridge Uni. Press, 2009.

[12] T. W. Frühwirth. Constraint handling rules - what else? In N. Bassiliades, G. Gottlob, F. Sadri, A. Paschke, and D. Roman, editors, *Rule Technologies: Foundations, Tools, and Applications - 9th International Symposium, RuleML 2015, Berlin, Germany, August 2-5, 2015, Proceedings*, volume 9202 of *Lecture Notes in Computer Science*, pages 13–34. Springer, 2015.

[13] D. Gall and T. Frühwirth. Confluence modulo equivalence with invariants in Constraint Handling Rules. In *FLOPS 2018*, 2018. To appear.

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

[15] M. H. Kirkeby and H. Christiansen. Confluence of CHR revisited: invariants and modulo equivalence. *CoRR*, 2018. *(submitted to an international symposium)*.

[16] J. Langbein, F. Raiser, and T. W. Frühwirth. A state equivalence and confluence checker for CHRs. In *Proc. Int'l Workshop on Constraint Handling Rules*, Report CW 588, pages 1–8. Katholieke Universiteit Leuven, Belgium, 2010.

Critical pairs for Gray categories

Simon Forest

Ecole Polytechnique

Abstract

Higher categories are a generalization of standard categories where there are not only 1-cells between 0-cells but more generally n+1-cells between n-cells. Semi-strict categories, such as Gray categories in dimension 3, is a flavour of higher categories suited for rewriting and used in this work. Here, we are interested in proving *coherence* of certain algebraic structures in dimension 3 using rewriting, where "coherence" is the property that there is at most one 3-cell between two 2-cells. Checking coherence then amounts to compute critical pairs of a rewriting system and use a variant of Newmann's lemma. In this setting, an algorithm exists to compute these critical pairs.

Introduction

It is well-known that rewriting can be used to manipulate algebraic theories. In this setting, the terms are the algebraic terms that arise from the *signature* of the theory and rewrite rules come from an orientation of the equations of the theory. In the context of higher categories, these techniques need to be adapted. Take monoids as an example. A monoid is given by a set M, an operation $m : M \times M \to M$ and an element $e \in M$ such that $m(m(x,y),z) = m(x,m(y,z))$, $m(x,e) = x = m(e,x)$. More generally, there is a notion of monoid in 2-category where the elements m and e are 2-generators in a 2-category: $m : M *_0 M \Rightarrow M$ and $e : 1 \Rightarrow M$ and such that equalities of 2-cells similar to the previous ones hold. The term rewriting system (or *TRS*) associated to the theory of monoids is then given by the signature $\mathsf{S} = \{m : M \times M \to M, e : 1 \to M\}$ and the following rewrite rules on formal compositions obtained by orienting the equations: $m \circ (m \times 1_M) \to m \circ (1_M \times m)$, $m \circ (e \times 1_M) \to 1_M$ and $m \circ (1_M \times e) \to 1_M$. The standard tools of rewriting i.e., termination, critical pair lemma and Newman's lemma entails uniqueness of normal forms. In order to go from interpretations in n-categories to interpretations in n+1-categories, the usual recipe is to replace equations on n-cells by n+1-isomorphisms and by adding equations on the new n+1-cells, called *coherence cells*, in order to entail the property of *coherence*, which states that, modulo the equations, there is at most one 3-cell between two 2-cells. For monoids, by going from dimension 2 to 3, we obtain the theory of pseudomonoids, which is important in category theory since the notion of monoidal category can be seen as a pseudomonoid in the category of categories.

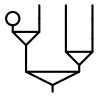
Several variants of 3-categories exist with different levels of expressivity and ease to manipulate. On the one end of the spectrum, weak 3-categories are the most general but are complex since they have a lot of coherence cells. On the other end, strict 3-categories have no coherence cells, only simple equations. But they are less expressive. Gray categories [4, 5] are a middle ground between the two. In this work, we will study interpretations of algebraic structures inside Gray categories. As a previous work[3] has shown, in order to have coherence, it is sufficient to enforce equations on coherence cells that come from the *critical branchings* (or critical pairs) of an adequate rewriting system. So there is a strong need for a tool that can automate the computation of these critical branchings.

1 Signatures and rewriting system

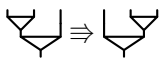
A *graph* $(\mathsf{S}_0, \mathsf{S}_1, \mathsf{s}_0, \mathsf{t}_0)$ is given by a set S_0 of *points* and a set S_1 of *arrows* and source and target functions $\mathsf{s}_0, \mathsf{t}_0 : \mathsf{S}_1 \to \mathsf{S}_0$. We denote S_1^* the set of paths in the graph and $\mathsf{s}_0^*, \mathsf{t}_0^* : \mathsf{S}_1^* \to \mathsf{S}_0$ the source and the target functions on paths, and $*_0$ the composition operation on paths. A *signature* S is given by a graph $(\mathsf{S}_0, \mathsf{S}_1, \mathsf{s}_0, \mathsf{t}_0)$, by a set of 2-generators S_2 with source and target functions $\mathsf{s}_1, \mathsf{t}_1 : \mathsf{S}_2 \to \mathsf{S}_1^*$ such that $\mathsf{s}_0^* \circ \mathsf{s}_1 = \mathsf{s}_0^* \circ \mathsf{t}_1$ and $\mathsf{t}_0^* \circ \mathsf{s}_1 = \mathsf{t}_0^* \circ \mathsf{t}_1$. An example of signature is the *monoid* signature P, where:

$$\mathsf{P}_0 = \{\star\} \qquad \mathsf{P}_1 = \{1 : \star \to \star\} \qquad \mathsf{P}_2 = \{\mu : 2 \Rightarrow 1, \eta : 0 \Rightarrow 1\}$$

Note that we write n for the path $\underbrace{\star \xrightarrow{1} \star ... \star \xrightarrow{1} \star}_{n}$. A *whisker* w is then given by two paths $u, v \in \mathsf{S}_1^*$ and a 2-generator $\alpha \in \mathsf{S}_2$ and is denoted $u * \alpha * v$. The 1-*source* and the 1-*target* are defined as $u *_0 \mathsf{s}_1\alpha *_0 v$ and $v *_0 \mathsf{t}_1\alpha *_0 v$ and are respectively denoted $\mathsf{s}_1 w$ and $\mathsf{t}_1 w$. A 2-*cell* α is given by a sequence of whiskers $w_1, ..., w_p$ that are 1-*composable*, i.e., $\mathsf{t}_1 w_i = \mathsf{s}_1 w_{i+1}$. We denote α as $w_1 * ... * w_p$. The 1-*source* and the 1-*target* of α are defined as $\mathsf{s}_1 w_1$ and $\mathsf{t}_1 w_p$ and are denoted $\mathsf{s}_1\alpha$ and $\mathsf{t}_1\alpha$ respectively. We denote S_2^* the set of 2-cells. For two 1-composable cells $\alpha = w_1 * ... * w_p$ and $\beta = w'_1 * ... * w'_q$, we define the 1-composition $\alpha *_1 \beta$ as the 2-cell $w_1 * ... * w_p * w'_1 * ... * w'_q$. Note that 2-cells can easily be represented as string diagrams. For example, in the case of monoids, if we picture η with  and μ with  , the following two 2-cells can be defined:

$(0 * \eta * 3) * (0 * \mu * 2) * (1 * \mu * 0) * \mu =$  $(0 * \eta * 3) * (2 * \mu * 0) * (0 * \mu * 1) * \mu =$ 

Note that in these pictures, there can be only one generator at a given height, and the relative heights matter, so that the two 2-cells are not considered to be equal (contrarily to 2-categories). A *rewriting system* consists of a signature S together with a set S_3 of 3-*generators*, or *rewriting rules*, equipped with source and target functions $\mathsf{s}_2, \mathsf{t}_2 : \mathsf{S}_3 \to \mathsf{S}_2^*$. For example, the rewriting system of monoids, which extends the associated signature, has the following rewrite rules:

$A : (\mu * 1) * \mu \Rrightarrow (1 * \mu) * \mu$  $L : (\eta * 1) * \mu \Rrightarrow \mu$  $R : (1 * \eta) * \mu \Rrightarrow \mu$ 

A *context* $E = \phi * (u * \_ * v) * \psi$ is given by $u, v \in \mathsf{S}_1^*$ and $\phi, \psi \in \mathsf{S}_2^*$. For A a rewrite rule, E is *compatible with* θ when $E[A] = \phi *_1 (u *_0 A *_0 v) *_1 \psi$ exists. A *rewriting step* R is then given by a rewrite rule $A \in \mathsf{S}_3$ and a compatible context E. It can be seen as an elementary 3-cell of type $\phi *_1 (u *_0 \mathsf{s}_2 A *_0 v) *_1 \psi \Rrightarrow \phi *_1 (u *_0 \mathsf{t}_2 A *_0 v) *_1 \psi$. A *rewriting path* is a finite sequence of composable rewriting steps $R_1, ..., R_n$ with $R_i : \theta_i \Rrightarrow \theta_{i+1}$. We denote such a rewriting path as $R_1 * ... * R_n$ or 1_θ for the empty path starting from the 2-cell θ. We write S_3^* for the set of rewriting paths, and $\mathsf{s}_2^*, \mathsf{t}_2^* : \mathsf{S}_3^* \to \mathsf{S}_2^*$ for the associated source and target functions. If $R : \theta_1 \Rrightarrow \theta_2$ is a rewrite step, we define the *reverted rewrite step* $R^{-1} : \theta_2 \Rrightarrow \theta_1$ as a formal inverse of R. Then, a *rewriting zigzag* is a sequence $Z_1, ..., Z_n$ where Z_i is either a rewrite step or a reverted rewrite step and such that $\mathsf{t}_2 Z_i = \mathsf{s}_2 Z_{i+1}$. We denote such a rewriting zigzag as $Z_1 * ... * Z_n$. We denote $\mathsf{S}_3^\top$ the set of rewriting zigzags. If $Z = Z_1 * ... * Z_n$, we define Z^{-1} as the zigzag $Z' = Z'_n * ... * Z'_1$ with $Z'_i = R^{-1}$ if $Z_i = R$ and $Z'_i = R$ if $Z_i = R^{-1}$. There is also a composition operation of zigzags given by the concatenation of sequences. A *coherated rewriting system* is given by a

rewrite system S and a congruence $\equiv$ on the rewriting zigzags $\mathsf{S}_3^\top$. By "congruence", we mean that $\equiv$ is an equivalence relation compatible with the difference compositions and the inverse operations. For example, if $Z_1 \equiv Z_2$ then $U * Z_1 * V \equiv U * Z_2 * V$ and ${Z_1}^{-1} \equiv {Z_2}^{-1}$. The *standard congruence* $\equiv_S$ on a rewriting system S is the smallest congruence such that:

1. if $E_1[A_1]$ and $E_2[A_2]$ are two rewrite steps of same source θ and "acting on independent zones of θ" (notion that will be precised later) then $E_1[A_1] * E_2'[A_2] \equiv_S E_2[A_2] * E_1'[A_1]$ where E_1' is the "residual context" of E_1 after the rewrite step $E_2[A_2]$ and similarly for E_2'.
2. if R is a rewrite step, then $R * R^{-1} \equiv_S 1_{\mathsf{s}_2 R}$ and $R^{-1} * R \equiv_S 1_{\mathsf{t}_2 R}$

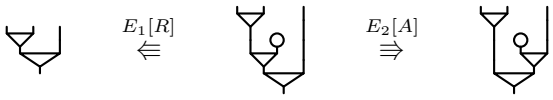
For instance, in the rewriting system of monoids, there is the following instance of 1:

We say that a congruence $\equiv$ is *standard* when $\equiv_S \subset \equiv$. Note that signatures and rewriting systems are simplified definitions of *prepolygraphs*, or *polygraphs on precategories* [6, 3].

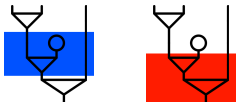
In what follows, we supposed a fixed rewriting system S.

2 Rewriting

Branchings. A *branching* $\mathcal{B}$ is a pair of rewriting steps $(R_1, R_2) = (E_1[A_1], E_2[A_2])$ with $\mathsf{s}_2 R_1 = \mathsf{s}_2 R_2$. We call the *source of the branching* $\mathcal{B}$, denoted $\mathsf{s}_2\mathcal{B}$ to be $\mathsf{s}_2 R_1$. Recall that a 2-cell θ is of the form $w_1 * ... * w_n$. We then define the *size* of θ, denoted $|\theta|$, as n and the *interval* of θ, denoted I_θ, as the set $\{1, ..., n\}$. If A is a rewrite rule and $E = \phi * (u * \_ * v) * \psi$ is a compatible context, we define the *action interval* of $E[A] : \theta \Rrightarrow \theta'$ on θ, denoted $I_{E[A]}$ to be the subsegment $\{|\phi| + 1, ...|\phi| + |\mathsf{s}_2 A|\}$ of I_θ and the *action index* to be $|\phi|$. Note that the size of $I_{E[A]}$ is given by $|\mathsf{s}_2 A|$. In what follows, we will suppose that for all rewrite rules $A \in \mathsf{S}_3$, we have $|\mathsf{s}_2 A| \geq 1$. For a branching $\mathcal{B} = (E_1[A_1], E_2[A_2])$ with $E_i = \phi_i * (u_i * \_ * v_i) * \psi_i$, define the *relative offset* of $\mathcal{B}$ to be $|\phi_2| - |\phi_1|$. Also, we say that that the actions of $E_1[A_1]$ and $E_2[A_2]$ are *overlapping* if $I_{E_1[A_1]} \cap I_{E_2[A_2]} \neq \emptyset$. For example, in the theory of monoids, there is the following branching:



whose action indexes are respectively 1 and 2 and whose action intervals are respectively $\{2, 3\}$ and $\{3, 4\}$, are overlapping and can be depicted as follows:



Let $\mathcal{B} = (E_1[A_1], E_2[A_2])$ a branching with $E_i = \phi_i * (u_i * \_ * v_i) * \psi_i$. We say that $\mathcal{B}$ is *trivial* when $E_1[A_1] = E_2[A_2]$, *non-minimal* when there is another branching $(F_1[A_1], F_2[A_2])$ with $F_i = \alpha_i * (r_i * \_ * s_i) * \beta_i$ such that there exists r, s, α, β not all identities such that $\phi_i = \alpha * \alpha_i$, $\psi_i = \beta_i * \beta$, $u_i = r * r_i$ and $v_i = s_i * s$, *independent* when the actions $E_1[A_1]$ and $E_2[A_2]$ are not

overlapping and *critical* when it is of none of the above. We adapt the notion of confluence in the setting of a coherated rewriting system $(\mathsf{S}, \equiv)$: a branching $\mathcal{B} = (R_1, R_2)$ with $R_i : \alpha \Rrightarrow \beta_i$ is said to be *confluent* when there exists a pair of rewriting paths (S_1, S_2) with $S_i : \beta_i \Rrightarrow \gamma$ such that $R_1 * S_1 \equiv R_2 * S_2$. We also define a straight-forward notion of termination: we say that a rewriting system S is *terminating* if there is no infinite sequence $(R_i)_{i\in\mathbb{N}}$ of rewriting steps with $R_i : \phi_i \Rrightarrow \phi_{i+1}$. In another work, we have the following result that motivates the computing of critical branchings:

Theorem 1 (FSCD18, Newman's lemma)**.** *Let $(\mathsf{S}, \equiv)$ be a coherated rewriting system such that the rewriting system S is terminating, $\equiv$ is standard and all critical branchings are confluent. Then $(\mathsf{S}, \equiv)$ is coherent.*

Computation of critical branchings. Let $\mathcal{B} = (E_1[A_1], E_2[A_2])$ be a critical branching. Because $\mathcal{B}$ is in particular non-independent, it holds that $I_{E_1[A_1]} \cap I_{E_2[A_2]} \neq \emptyset$. But since $I_{E_i[A_i]} = \{|\phi_i| + 1, ..., |\phi_i| + |\mathsf{s}_2 A_i|\}$, the relative offset is bounded: $0 \leq |\phi_2| - |\phi_1| < |\mathsf{s}_2 A_1|$. Moreover, for a given offset, we have a uniqueness property:

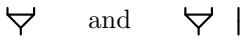
Proposition 1. *Let A_1 and A_2 be two rewrite rules and p such that $0 \leq p < |\mathsf{s}_2 A_1|$. Then there is at most one critical branching $\mathcal{B} = (E_1[A_1], E_2[A_2])$ such that the actions of $E_1[A_1]$ and $E_2[A_2]$ have p as relative offset.*

Let A_1 and A_2 and n_1, n_2 such that $n_i = |\mathsf{s}_2 A_i|$ and

$$\mathsf{s}_2 A_i = (r_{i,1} * \alpha_{i,1} * s_{i,1}) * ... * (r_{i,n_1} * \alpha_{i,n_1} *_0 s_{i,n_1})$$

The proof of the last property gives us a procedure to compute all the context E_1, E_2 such that $(E_1[A_1], E_2[A_2])$ is a critical branching. See figure 2 for the procedure.

Example. We apply this procedure for the computation of critical branchings between the rewrite rules A and A in the theory of monoids. There are only two possible relative offset p to test: 0 and 1. When $p = 0$, the procedure produces no context because it is the case of the trivial branching. So we focus on the case $p = 1$. In this case, the two whiskers to unify are the following:



It is easy to unify them using $u_1 = 0$, $u_2 = 0$, $v_1 = 1$ and $v_2 = 0$. Using the formulas of the procedure, we then define

These elements define contexts E_1, E_2 with $E_i = \phi_i * (u_i * \_ * v_i) * \psi_i$ and they define a branching $\mathcal{B} = (E_1[A], E_2[A])$ where $E_1[A]$ and $E_2[A]$ are rewrite step as follows:

$\mathcal{B}$ is non-independent since the action intervals of $E_1[A]$ and $E_2[A]$ are respectively $\{1, 2\}$ and $\{2, 3\}$ so they overlap. Moreover, this branching is minimal. So $\mathcal{B}$ is critical.

procedure CRITICALBRANCHING(A_1, A_2)
  let $P = \emptyset$
  for $p = 0$ **to** $n_1 - 1$ **do** (MainFor) ▷ All possible relative offset are tested
    if ($p = 0$ **and** $A_1 = A_2$)
     or ($r_{1,p+1}$ is not a suffix of $r_{2,1}$ **and** $r_{2,1}$ is not a suffix of $r_{1,p+1}$)
     or ($s_{1,p+1}$ is not a prefix of $s_{2,1}$ **and** $s_{2,1}$ is not a prefix of $s_{1,p+1}$) **then**
      continue MainFor
    end if
    let u_1, u_2 **be** the smallest such that $u_1 * r_{1,p+1} = u_2 * r_{2,1}$
    let v_1, v_2 **be** the smallest such that $s_{1,p+1} * v_1 = s_{2,1} * v_2$
    for $i = p + 1$ **to** n_1 **do**
      if $u_1 * r_{1,i} \neq u_2 * r_{2,i}$ **or** $\alpha_{1,i} \neq \alpha_{2,i-p}$ **then**
        continue MainFor
      end if
    end for
    let $\phi_1 = 1$ **and** $\phi_2 = *_{i=1}^{p}((u_1 * r_{1,i}) * \alpha_{1,i} * (s_{1,i} * v_1))$
    let $\psi_1 = *_{i=n_1-p+1}^{n_2}((u_2 * r_{2,i}) * \alpha_{2,i} * (s_{2,i} * v_2))$
    and $\psi_2 = *_{i=n_2+p+1}^{n_1}((u_1 * r_{1,i}) * \alpha_{1,i} * (s_{1,i} * v_1))$
    $P \leftarrow P \cup \{(\phi_1 * (u_1 * \_ * v_2) * \psi_1, \phi_2 * (u_2 * \_ * v_2) * \psi_2)\}$
  end for
  return P
end procedure

Conclusion

In this work, we showed how rewriting formalism can be used for the interpretation of algebraic structures in a 3-dimensional categorical setting. In particular, we defined the notion of signatures, rewriting systems, rewrite rules and rewrite paths in this setting and stated an adaptation of Newman's lemma for coherence, which relates the coherence property to the critical branchings of the rewriting system. Then we gave an algorithm to compute the critical branchings, and gave an example for the theory of pseudomonoids. Even though we restricted ourselves to dimension 3, the formalism and the algorithm can be readily used with higher dimensions.

References

[1] Krzysztof Bar, Aleks Kissinger, and Jamie Vicary. Globular: an online proof assistant for higher-dimensional rewriting. In *LIPIcs*, volume 52, pages 34:1–34:11, 2016.

[2] Krzysztof Bar and Jamie Vicary. Data structures for quasistrict higher categories. In *Logic in Computer Science (LICS), 32nd Annual Symposium on*, pages 1–12. IEEE, 2017.

[3] Simon Forest and Samuel Mimram. Coherence of Gray categories via rewriting. Submitted, 2018.

[4] Robert Gordon, Anthony John Power, and Ross Street. *Coherence for tricategories*, volume 558. American Mathematical Soc., 1995.

[5] Michael Nicholas Gurski. *An algebraic theory of tricategories.* PhD thesis, University of Chicago, Department of Mathematics, 2006.

[6] Aleks Kissinger and Jamie Vicary. Semistrict n-categories via rewriting. Proceedings of the first workshop on *Higher-Dimensional Rewriting and Applications*, 2015.

[7] Dominic Verdon. Coherence for braided and symmetric pseudomonoids. Preprint, 2017.

Coherence of monoids by insertions

Nohra Hage[1] and Philippe Malbos[2]

[1] Ecole supérieure d'ingénieurs de Beyrouth (ESIB), USJ, Liban, nohra.hage@usj.edu.lb
[2] Institut Camille Jordan, Université de Lyon, France, malbos@math.univ-lyon1.fr

Abstract

We introduce string data structures as combinatorial descriptions of structured words on totally ordered alphabets. The data can be described by words through a reading map and can be constructed by using an insertion algorithm. The insertion map defines a product on datum. We show that the associativity of this product, the cross-section property of the data structure, and the confluence of the rewriting system defined by the insertion map are equivalent properties. We make explicit a coherent presentation of the monoid presented by the data structure, made of generators, rewriting rules describing the insertion of letters in words and relations among the insertion algorithms.

1 Introduction

A data structure describes a way to organize and to store a collection of structured data. It defines primitive operations such as constructors, insertion and reading maps on the data. In this work, we study string rewriting systems (SRS) whose normal forms can be described using a data structure and whose normalisation strategies are induced by insertion algorithms. Such data structures appear in many contexts in combinatorial algebra, combinatorics and fundamental computer science and describe combinatorial structures such as arrays, tableaux, staircases or binary search trees. They are used to describe combinatorially equivalence relations in free monoids. In particular array structures can be used to study plactic, Chinese, hypoplactic and sylvester monoids.

For instance, the structure of plactic monoid emerged from the works of Schensted [13] and Knuth [9] on the combinatorial study of Young tableaux and it has found several applications in combinatorics and representation theory [11, 4]. The study of plactic monoids (of type A) using SRS on Knuth generators is not straightforward, in particular in rank greater than 4 they do not admit finite completion with respect to the lexicographic order, [10]. Finite completions can be obtained by adding new generators in the quasi-center of the monoid. In particular, by adding column or row generators, the completion procedure ends producing a convergent presentation of plactic monoids, [2, 1]. Such convergent presentations can be used to make explicit coherent presentations of plactic monoids giving all the relations among the relations of the presentations, [7]. The confluence property is essential to obtain such coherence results.

The confluence of the column presentation for plactic monoids is a consequence of the commutation of Schensted's insertion algorithms in Young tableaux: the right insertion (or insertion by rows) and the left insertion (or insertion by columns). In this work, we make explicit this confluence result in a general algebraic framework. We introduce the notion of string data structures as combinatorial descriptions of structured words on totally ordered alphabets. The data can be described by words through a reading map and can be constructed using an insertion algorithm. The insertion map defines a product on datum. We show that the associativity of this product, the cross-section property of the data structure, and the confluence of the rewriting system defined by the insertion map are equivalent properties. We make explicit a coherent presentation of the monoid presented by the data structure, made of generators, rewriting rules describing the insertion of letters in words and relations among the insertion algorithms.

In a first part, we introduce the notion of string data structure. We show that the commutation of left and right insertion algorithms on a data structure induces an associative product on the data. We define an SRS associated to a data structure, whose rules are defined by the insertion map, and we show that the associativity of the product on the data structure yields the confluence of this SRS. In a second part, using the notion of generating set of a string data structure, we construct an SRS on a reduced set of data and we show that the associativity of the data structure induces the confluence of this reduced SRS. In addition, we make explicit a coherent presentation of the monoid presented by a data structure in terms of the normalisation strategy induced by the insertion algorithm on the data structure. We recall in Appendices the Schensted's algorithms, the notion of coherent presentation and we give the proofs of the main results presented in this abstract.

2 String data structures, confluence and cross-section

String data structures. A *string data structure*, SDS for short, $\mathbb{S}$ on a totally ordered alphabet A is a quadruple $(\mathrm{D}_A, \ell, I, R)$ made of a set D_A, a reading ℓ of words on A, a *one-element insertion map* I and a *reading map* R defined as follows:

i) the inclusions $A \subseteq R(\mathrm{D}_A) \subseteq A^*$ hold, where A^* denotes the free monoid on A,

ii) the map $\ell : A^* \to A^*$ sends each word $x_1 \dots x_k$ in A^* on a word $x_{\sigma_{(1)}} \dots x_{\sigma_{(k)}}$ in A^*, where σ is a permutation on $\{1, \dots, k\}$,

iii) $I : \mathrm{D}_A \times A \to \mathrm{D}_A$ inserts an element of A into an element of D_A such that any restriction $I(-, x)$ is injective for $x \in A$. By iteration, one defines an *insertion map* $I^* : \mathrm{D}_A \times A^* \to \mathrm{D}_A$ that inserts a word in A^* into an element of D_A wrt ℓ, that is $I^*(d, x_1 \dots x_n) = I^*(I(d, y_1), y_2 \dots y_n)$, for any $d \in \mathrm{D}_A$ and $x_1 \dots x_n \in A^*$, where $y_1 \dots y_n = \ell(x_1 \dots x_n)$,

iv) $R : \mathrm{D}_A \to A^*$ is injective and satisfies $I^*(\emptyset, \ell(-))R = \mathrm{Id}_{\mathrm{D}_A}$ and $R(\emptyset)$ is the empty word.

The map $I^*(\emptyset, \ell(-)) : A^* \to \mathrm{D}_A$ is called the *constructor* of the SDS $\mathbb{S}$. The maps I, R and $I^*(\emptyset, \ell(-))$ will be also denoted by $I_{\mathbb{S}}$, $R_{\mathbb{S}}$ and $C_{\mathbb{S}}$. We will use the *right-to-left* (resp. *left-to-right*) reading of words denoted by ℓ_{r} (resp. ℓ_{l}). A *right* (resp. *left*) *SDS* is an SDS whose insertion map is said *right* (resp. *left*), that is inserting a word into an element of D_A with respect to ℓ_{l} (resp. ℓ_{r}). Two one-element insertion maps $I, J : \mathrm{D}_A \times A \to \mathrm{D}_A$ *commute* if the relation $J(I(d, x), y) = I(J(d, y), x)$ holds for every $d \in \mathrm{D}_A$ and $x, y \in A$. An *opposite* of a right (resp. left) SDS $(\mathrm{D}_A, \ell_{\mathrm{l}}, I, R)$ (resp. $(\mathrm{D}_A, \ell_{\mathrm{r}}, I, R)$) is a left (resp. right) SDS $(\mathrm{D}_A, \ell_{\mathrm{r}}, J, R)$ (resp. $(\mathrm{D}_A, \ell_{\mathrm{l}}, J, R)$) such that I and J commute.

For example, a *(Young) tableau* on the finite set $[n] := \{1, \dots, n\}$ is a collection of boxes in left-justified rows, filled with elements of $[n]$, where the entries weakly increase along each row and strictly increase down each column. Denote by Yt_n the set of tableaux on $[n]$. Schensted, [13], introduced the *right (or row)* (resp. *left (or column)* insertion S_r (resp. S_l) : $\mathrm{Yt}_n \times [n] \to \mathrm{Yt}_n$, see Appendix A. Let $R_{col} : \mathrm{Yt}_n \to [n]^*$ be the map reading the columns of a tableau from left to right and from bottom to top. This defines two SDSs $\mathcal{Y}_n^{row} = (\mathrm{Yt}_n, \ell_{\mathrm{l}}, S_r, R_{col})$ and $\mathcal{Y}_n^{col} = (\mathrm{Yt}_n, \ell_{\mathrm{r}}, S_l, R_{col})$ on the structure of tableau.

An SDS $\mathbb{S} = (\mathrm{D}_A, \ell, I, R)$ is *associative* if the product $\star_{\mathbb{S}} : \mathrm{D}_A \times \mathrm{D}_A \to \mathrm{D}_A$ defined by setting $d \star_{\mathbb{S}} d' = I^*(d, \ell(R(d')))$, for any $d, d' \in \mathrm{D}_A$ is associative. That is, the relation $(d \star_{\mathbb{S}} d') \star_{\mathbb{S}} d'' = d \star_{\mathbb{S}} (d' \star_{\mathbb{S}} d'')$ holds for any $d, d', d'' \in \mathrm{D}_A$. For instance, the SDS $(\mathrm{Yt}_n, \ell_{\mathrm{l}}, S_l, R_{col})$ is not associative:

$$\left(\begin{array}{|c|}\hline 1\\\hline 4\\\hline 6\\\hline\end{array} \star_{\mathbb{S}} \begin{array}{|c|}\hline 2\\\hline 3\\\hline\end{array}\right) \star_{\mathbb{S}} \boxed{1} = \begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4\\\cline{1-1} 6\\\cline{1-1}\end{array} \star_{\mathbb{S}} \boxed{1} = \begin{array}{|c|c|c|}\hline 1&1&3\\\hline 2\\\cline{1-1} 4\\\cline{1-1} 6\\\cline{1-1}\end{array} \neq \begin{array}{|c|c|c|c|}\hline 1&1&2&3\\\hline 4\\\cline{1-1} 6\\\cline{1-1}\end{array} = \begin{array}{|c|}\hline 1\\\hline 4\\\hline 6\\\hline\end{array} \star_{\mathbb{S}} \begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array} = \begin{array}{|c|}\hline 1\\\hline 4\\\hline 6\\\hline\end{array} \star_{\mathbb{S}} \left(\begin{array}{|c|}\hline 2\\\hline 3\\\hline\end{array} \star_{\mathbb{S}} \boxed{1}\right)$$

Theorem 1. *Let $\mathbb{S}$ be a right (resp. left) SDS. If there is a left (resp. right) SDS $\mathbb{T}$ opposite to $\mathbb{S}$, then the SDSs $\mathbb{S}$ and $\mathbb{T}$ commute, that is $d \star_{\mathbb{S}} d' = d' \star_{\mathbb{T}} d$, for any $d, d' \in \mathrm{D}_A$, and are associative.*

Structure monoid of an SDS. Let $\mathbb{S} = (\mathrm{D}_A, \ell, I, R)$ be an SDS. Denote by $|$ the product of the free monoid on D_A. The *structure monoid* associated to the SDS $\mathbb{S}$ is the monoid, denoted by $\mathbf{M}(\mathbb{S})$, and presented by the following SRS

$$\mathcal{R}(\mathbb{S}) = \langle\, \mathrm{D}_A \mid \gamma_{d,d'} : d|d' \to d \star_{\mathbb{S}} d' \text{ for any } d, d' \text{ in } \mathrm{D}_A \,\rangle,$$

called the *standard presentation* induced by the SDS $\mathbb{S}$. Since every application of a rewriting rule of $\mathcal{R}(\mathbb{S})$ yields a strictly smaller preceding word with respect to the deglex order on D_A^*, the SRS $\mathcal{R}(\mathbb{S})$ is terminating. Moreover, if the SDS $\mathbb{S}$ is associative, then the SRS $\mathcal{R}(\mathbb{S})$ is convergent. The *reading of the standard presentation* of the SDS $\mathbb{S}$ is the SRS defined by

$$\mathcal{R}(A, \mathbb{S}) = \langle\, A \mid \gamma_{d,d'} : R_{\mathbb{S}}(d)R_{\mathbb{S}}(d') \to R_{\mathbb{S}}(d \star_{\mathbb{S}} d') \text{ for any } d, d' \text{ in } \mathrm{D}_A \,\rangle.$$

Any critical pair of $\mathcal{R}(A, \mathbb{S})$ has the form

$$\begin{array}{ccccc} & \xrightarrow{\gamma_{d,d'}R_{\mathbb{S}}(d'')} & R_{\mathbb{S}}(d \star_{\mathbb{S}} d')R_{\mathbb{S}}(d'') & \xrightarrow{\gamma_{d\star_{\mathbb{S}}d',d''}} & R_{\mathbb{S}}((d \star_{\mathbb{S}} d') \star_{\mathbb{S}} d'') \\ R_{\mathbb{S}}(d)R_{\mathbb{S}}(d')R_{\mathbb{S}}(d'') & & & & \\ & \xrightarrow[R_{\mathbb{S}}(d)\gamma_{d',d''}]{} & R_{\mathbb{S}}(d)R_{\mathbb{S}}(d' \star_{\mathbb{S}} d'') & \xrightarrow[\gamma_{d,d'\star_{\mathbb{S}}d''}]{} & R_{\mathbb{S}}(d \star_{\mathbb{S}} (d' \star_{\mathbb{S}} d'')) \end{array}$$

for every $d, d', d'' \in \mathrm{D}_A$. As a consequence, if $\mathbb{S}$ is associative, then $\mathcal{R}(A, \mathbb{S})$ is locally confluent.

Compatibility of an SDS. An associative SDS $\mathbb{S} = (\mathrm{D}_A, \ell, I, R)$ is *compatible* with an equivalence relation $\sim$ on A^* if for any $d \in \mathrm{D}_A$ and $w, w' \in A^*$, $w \sim w'$ implies $I^*(d, w) = I^*(d, w')$, and for any $w \in A^*$, one has $R_{\mathbb{S}}C_{\mathbb{S}}(w) \sim w$. If $\mathbb{S} = (\mathrm{D}_A, \ell, I, R)$ is an associative SDS compatible with the relation $\sim$, thus I^* induces a unique map $\widetilde{I^*}$, such that the diagram on the right commutes, where $\pi : A^* \to A^*/\sim$ denotes the quotient map. Hence, the constructor $C_{\mathbb{S}}$ induces a map $\overline{C_{\mathbb{S}}} : A^*/\sim \to \mathrm{D}_A$ defined by $\overline{C_{\mathbb{S}}}(\pi(w)) = \widetilde{I^*}(\emptyset, \pi(\ell(w)))$, for any $w \in A^*$. Moreover, we have $\overline{C_{\mathbb{S}}}\pi R_{\mathbb{S}} = Id_{\mathrm{D}_A}$. Hence, the map $\overline{C_{\mathbb{S}}}$ is bijective.

$$\begin{array}{ccc} \mathrm{D}_A \times A^* & \xrightarrow{I^*} & \mathrm{D}_A \\ {\scriptstyle \mathrm{Id}\times\pi}\downarrow & \nearrow_{\widetilde{I^*}} & \\ \mathrm{D}_A \times A^*/\sim & & \end{array}$$

Proposition 1. *Let $\mathbb{S}$ be a right associative SDS compatible with the equivalence relation $\sim_{\mathbb{S}}$ induced by $\mathcal{R}(A, \mathbb{S})$. The map $C_{\mathbb{S}} : A^* \to \mathrm{D}_A$ induces a monoid isomorphism $\overline{C_{\mathbb{S}}}$ between $A^*/\sim_{\mathbb{S}}$ and $(\mathrm{D}_A, \star_{\mathbb{S}})$, with the inverse induced by the reading map $R_{\mathbb{S}}$.*

Let $\sim$ be an equivalence relation on a free monoid K^* on a set K. Recall that a subset $S \subset K^*$ *satisfies the cross-section property for the monoid $K^*/\sim$* if each equivalence class with respect to $\sim$ contains exactly one element of S. Let $\mathbb{S}$ be a right associative SDS compatible with the equivalence relation $\sim_{\mathbb{S}}$ induced by $\mathcal{R}(A, \mathbb{S})$. By Proposition 1, the monoids $(\mathrm{D}_A, \star_{\mathbb{S}})$ and $A^*/\sim_{\mathbb{S}}$ are isomorphic. One says that $\mathcal{R}(\mathbb{S})$ and $\mathcal{R}(A, \mathbb{S})$ are *Tietze-equivalent*, that is present the same monoid. In particular, if $\mathcal{R}(A, \mathbb{S})$ is terminating, then the set of normal forms wrt $\mathcal{R}(\mathbb{S})$ satisfies the cross-section property for $\mathbf{M}(\mathbb{S})$ if and only if the set $\mathrm{Nf}(A, \mathbb{S})$ of normal forms wrt $\mathcal{R}(A, \mathbb{S})$ satisfies the cross-section property for $\mathbf{M}(\mathbb{S})$.

Theorem 2. *Let $\mathbb{S}$ be a right associative SDS such that the SRS $\mathcal{R}(A, \mathbb{S})$ is terminating. Then the SRSs $\mathcal{R}(\mathbb{S})$ and $\mathcal{R}(A, \mathbb{S})$ are Tietze-equivalent and the set $\mathrm{Nf}(A, \mathbb{S})$ satisfies the cross-section property for the monoid $\mathbf{M}(\mathbb{S})$.*

For instance, the *plactic monoid* $\mathbf{P}_n$ of rank n, [11], is presented by the *Knuth presentation* whose set of generators is $[n]$ submitted to relations $zxy = xzy$ for $x \leqslant y < z$ and $yzx = yxz$ for $x < y \leqslant z$. Schensted showed that S_r and S_l commute, [13]. Then, by Theorem 1 the SDS $\mathcal{Y}_n^{row}$ is associative and the SRS $\mathcal{R}(\mathcal{Y}_n^{row})$ is convergent. One shows that the Knuth presentation is Tietze-equivalent to the reading of the SRS $\mathcal{R}([n], \mathcal{Y}_n^{row})$. By [9], see also [12], the SDS $\mathcal{Y}_n^{row}$ is compatible with the equivalence relation induced by the Knuth presentation. Then, by Proposition 1, $\mathcal{R}(\mathcal{Y}_n^{row})$ is a convergent presentation of the monoid $\mathbf{P}_n$. Hence, the set Yt_n satisfies the cross-section property for $\mathbf{P}_n$.

3 Coherent presentations and SDS

Change of generators. Let $\mathbb{S} = (\mathrm{D}_A, \ell, I, R_{\mathbb{S}})$ be an SDS. One considers a binary relation $|$ on D_A compatible with $R_{\mathbb{S}}$, that is $R_{\mathbb{S}}(d|d') = R_{\mathbb{S}}(d)R_{\mathbb{S}}(d')$ for any $d, d' \in \mathrm{D}_A$, where $d|d'$ denotes $(d, d') \in |$. A *generating set* with respect to such a binary relation is a subset Q of D_A such that $A \subseteq R_{\mathbb{S}}(Q)$, and any element d in D_A can be written $d = c_1|c_2|\dots|c_k$, with $c_1, \dots, c_k \in Q$. From such generating set Q of $\mathbb{S}$, one can define an SDS $\mathbb{S}_Q = (\mathrm{D}_A, \ell_Q, I_Q, R_Q)$ on Q, where
i) the map $\ell_Q : Q^* \to Q^*$ induces a permutation on the letters of each words on Q,
ii) $I_Q : \mathrm{D}_A \times Q \to \mathrm{D}_A$ is a one-element insertion map defined by $I_Q(d, c) = I^*(d, R_{\mathbb{S}}(c))$, for any $c \in Q$ and $d \in \mathrm{D}_A$, that induces an insertion map $I_Q^* : \mathrm{D}_A \times Q^* \to \mathrm{D}_A$ wrt ℓ_Q,
iii) $R_Q : \mathrm{D}_A \to Q^*$ is the reading map associated to $|$, that is, for any $d \in \mathrm{D}_A$, $R_Q(d) = c_1|c_2|\dots|c_k$ is the decomposition of d with respect to $|$.

A reduced presentation. Consider an SDS $\mathbb{S} = (\mathrm{D}_A, \ell, I, R_{\mathbb{S}})$ and a generating set Q of $\mathbb{S}$ with respect to a binary relation $|$ compatible with $R_{\mathbb{S}}$. One defines the following SRS

$$\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S}) = \langle\, Q \mid \gamma_{c,c'} : c|c' \to R_Q(c \star_{\mathbb{S}} c') \text{ for any } c, c' \in Q \text{ such that } c|c' \notin \mathrm{D}_A \,\rangle,$$

called the *reduced SRS* of $\mathbb{S}$. We will denote by $\mathrm{Nf}(Q, \mathbb{S})$ the set of normal forms wrt $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$. The SRS $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ may be non terminating, in particular when the number of generators in Q is not decreasing with the application of rules $\gamma_{c,c'}$. An additional condition is thus necessary on $\mathbb{S}$ to assure the termination of $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$.

Lemma 1. *Let $\mathbb{S} = (\mathrm{D}_A, \ell, I, R_{\mathbb{S}})$ be an associative SDS and Q be a generating set of $\mathbb{S}$ with respect to a binary relation $|$ compatible with $R_{\mathbb{S}}$. If the SRS $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ is terminating, then the SRSs $\mathcal{R}(\mathbb{S})$ and $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ are Tietze-equivalent.*

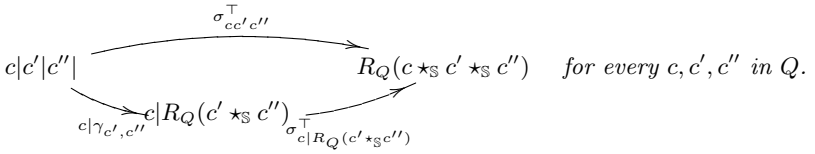
For instance, consider the SDS $\mathcal{Y}_n^{row}$ and let Col_n be the set of tableaux with only one column. Denote by $|$ the concatenation of columns in Yt_n. Every d in Yt_n can be written $d = c_1|c_2|\dots|c_k$, where $c_1, \dots, c_k$ are the columns of d from left to right. We have $R_{col}(d) = R_{col}(c_1)R_{col}(c_2)\dots R_{col}(c_k)$. Then, the concatenation $|$ is a binary relation compatible with R_{col} and the set Col_n is a generating set wrt $|$. Let $R_{\mathrm{Col}_n} : \mathrm{Yt}_n \to \mathrm{Col}_n^*$ be the map that decomposes a tableau as the concatenation of its columns from left to right. The SDS $\mathcal{Y}_n^{row}$ is associative and one shows that the SRS $\mathcal{R}(\mathrm{Col}_n, \mathrm{Yt}_n, \mathcal{Y}_n^{col})$ is terminating. Then by Lemma 1 the SRSs $\mathcal{R}(\mathcal{Y}_n^{col})$ and $\mathcal{R}(\mathrm{Col}_n, \mathrm{Yt}_n, \mathcal{Y}_n^{col})$ are Tietze-equivalent. Hence, the SRS $\mathcal{R}(\mathrm{Col}_n, \mathrm{Yt}_n, \mathcal{Y}_n^{col})$ is a finite convergent presentation of the monoid $\mathbf{P}_n$. By this way we recover the result of [1, 2].

Coherence from insertion. Recall that a *normalisation strategy* for an SRS R specifies a way to apply the rules in a deterministic way. It is defined as a mapping σ of every words u

in X^* to a rewriting step from u to a chosen normal form $\widehat{u}$. We distinguish two canonical strategies to reduce words: the *leftmost* one $\sigma^\top$ and the *rightmost* one $\sigma^\bot$, according to the way we apply first the rewriting rule that reduces the leftmost or the rightmost subword. Given an associative SDS $\mathbb{S} = (\mathrm{D}_A, \ell, I, R)$ and an associated reduced SRS $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$, we say that a normalization strategy σ of $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ *computes* the constructor $C_\mathbb{S}$ if it is normalizing, and it reduces any word $c_1|c_2|\dots|c_n$ in Q^* to $R_Q(c_1 \star_\mathbb{S} c_2 \star_\mathbb{S} \dots \star_\mathbb{S} c_n)$, that is

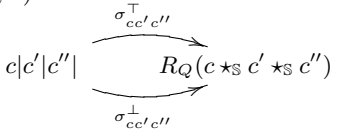
$$\sigma_{c_1|c_2|\dots|c_n} : c_1|c_2|\dots|c_n \to R_Q(c_1 \star_\mathbb{S} c_2 \star_\mathbb{S} \dots \star_\mathbb{S} c_n) \quad \text{for any } c_1, \dots, c_n \in Q.$$

Theorem 3. *Let $\mathbb{S}$ be an associative SDS and Q be a generating set of $\mathbb{S}$ such that the SRS $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ is terminating. If there exists a normalization strategy of $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ that computes $C_\mathbb{S}$, then the set $\mathrm{Nf}(Q, \mathbb{S})$ satisfies the cross-section property for $\mathbf{M}(\mathbb{S})$.*
In particular, if the leftmost normalization strategy $\sigma^\top$ computes $C_\mathbb{S}$, then the SRS $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ can be extended into a coherent convergent presentation by adjunction



for every c, c', c'' in Q.

With hypothesis of Theorem 3, consider $\sigma^\top$ (resp. $\sigma^\bot$) the leftmost (resp. rightmost) normalisation strategy with respect to $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ for a right SDS $\mathbb{S}$. Suppose that there is an SDS $\mathbb{T}$ opposite to $\mathbb{S}$. If the strategy $\sigma^\top$ computes $C_\mathbb{S}$, then $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ can be extended into a coherent convergent presentation by adjunction of the homotopy generator on the right for every c, c' and c'' in Q, where $\sigma^\top$ (resp. $\sigma^\bot$) corresponds to the application of the right (resp. left) insertion of $\mathbb{S}$ (resp. $\mathbb{T}$).



Theorem 3 can be used to construct coherent presentations of plactic monoids, see Appendix G.

4 Conclusion and work in progress

We have introduced the notion of SDS to study the confluence of SRS whose rules are defined by insertion algorithm. We show that the fact that a right SDS and a left SDS that present a monoid are opposite and the confluence property of the standard SRS presenting the monoid are equivalent properties. We apply our construction on the *Chinese monoid* of rank n, [3], generated by the set $[n]$ and subject to the relations $zyx = zxy = yzx$, for $x \leqslant y \leqslant z$. By constructing an SDS associated to the insertion algorithm in Chinese staircases, we deduce the confluence of the reduced presentation of the Chinese monoid and we extend this presentation into a finite coherent presentation of the monoid, see Appendix F. Finally, the *sylvester monoid* of rank n, [8], generated by $[n]$ and subject to the relations $cavb = acvb$, for all $a \leqslant b < c$ and $v \in [n]^*$, can be described using the notion of binary search trees. We expect that our methods should conduce to a coherent presentation of the sylvester monoid induced by the insertion algorithm in a binary search tree.

References

[1] L. Bokut, Y. Chen, W. Chen, and J. Li. New approaches to plactic monoid via Gröbner–Shirshov bases. *J. Algebra*, 423:301–317, 2015.

[2] A. J. Cain, R. D. Gray, and A. Malheiro. Finite Gröbner–Shirshov bases for Plactic algebras and biautomatic structures for Plactic monoids. *J. Algebra*, 423:37–53, 2015.

[3] J. Cassaigne, M. Espie, D. Krob, J.-C. Novelli, and F. Hivert. The Chinese monoid. *Int. J. Algebra Comput.*, 11(3):301–334, 2001.

[4] W. Fulton. *Young tableaux*, volume 35 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 1997. With applications to representation theory and geometry.

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

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

[7] N. Hage and P. Malbos. Knuth's coherent presentations of plactic monoids of type A. *Algebras and Representation Theory*, 20(5):1259–1288, Oct 2017.

[8] F. Hivert, J.-C. Novelli, and J.-Y. Thibon. The algebra of binary search trees. *Theoret. Comput. Sci.*, 339(1):129–165, 2005.

[9] D. E. Knuth. Permutations, matrices, and generalized Young tableaux. *Pacific J. Math.*, 34:709–727, 1970.

[10] Ł. Kubat and J. Okniński. Gröbner-Shirshov bases for plactic algebras. *Algebra Colloq.*, 21(4):591–596, 2014.

[11] A. Lascoux and M.-P. Schützenberger. Le monoïde plaxique. In *Noncommutative structures in algebra and geometric combinatorics (Naples, 1978)*, volume 109 of *Quad. "Ricerca Sci."*, pages 129–156. CNR, Rome, 1981.

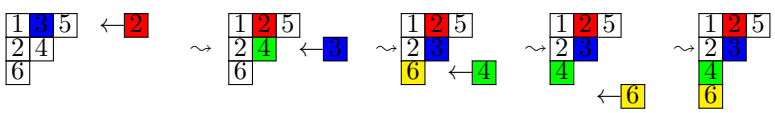
[12] V. Lebed. Plactic monoids: a braided approach. arXiv:1612.05768, 2016.

[13] C. Schensted. Longest increasing and decreasing subsequences. *Canad. J. Math.*, 13:179–191, 1961.

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

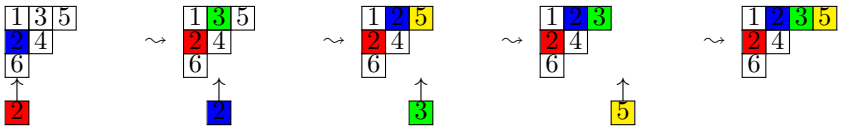
A Schensted's algorithms

Schensted introduced two algorithms to insert an element x in $[n]$ into a tableau t of Yt_n, [13]. The *right (or row) insertion algorithm* $S_r : \mathrm{Yt}_n \times [n] \longrightarrow \mathrm{Yt}_n$ computes a tableau $S_r(t,x)$ as follows. If x is at least as large as the last element of the top row of t, then put x to the right of this row. Otherwise, let y be the smallest element of the top row of t such that $y > x$. Then x replaces y in this row and y is bumped into the next row where the process is repeated. The procedure terminates when the element which is bumped is at least as large as the last element of the next row. Then it is placed at the right of that row. For instance, the four steps to compute $S_r\left(\begin{array}{|c|c|c|}\hline 1&3&5\\\hline 2&4\\\cline{1-2} 6\\\cline{1-1}\end{array}, 2\right)$ are:



The *left (or column) insertion algorithm* $S_l : \mathrm{Yt}_n \times [n] \longrightarrow \mathrm{Yt}_n$ computes a tableau $S_l(t,x)$ as follows. If x is larger than the first element of the first (leftmost) column of t, then put x to the

bottom of this column. Otherwise, let y be the smallest element of the first column of t such that $y \geqslant x$. Then x replaces y in this column and y is bumped into the next column where the process is repeated. The procedure terminates when the element which is bumped is greater than all the elements of the next column. Then it is placed at the bottom of that column. For instance, the four steps to compute $S_l\left(\begin{array}{|c|c|c|}\hline 1&3&5\\\hline 2&4\\\cline{1-2}6\\\cline{1-1}\end{array}, 2\right)$ are:



B Proof of Theorem 1

Consider a right SDS $\mathbb{S} = (\mathrm{D}_A, \ell_{\mathrm{l}}, I, R)$. Let $\mathbb{T} = (\mathrm{D}_A, \ell_{\mathrm{r}}, J, R)$ be an opposite SDS of $\mathbb{S}$. One shows that for any d, d' and d'' in D_A the following equalities hold

$$C_{\mathbb{S}}(R(d)R(d')R(d'')) = (d \star_{\mathbb{S}} d') \star_{\mathbb{S}} d'' \quad \text{and} \quad C_{\mathbb{T}}(R(d)R(d')R(d'')) = (d'' \star_{\mathbb{T}} d') \star_{\mathbb{T}} d. \tag{1}$$

Prove first that the equality $C_{\mathbb{S}}(w) = C_{\mathbb{T}}(w)$ holds for any w in A^*. We proceed by induction. By definition, we have $C_{\mathbb{S}}(x) = C_{\mathbb{T}}(x)$, for any x in A. Suppose that $C_{\mathbb{S}}(x_1 \dots x_k) = C_{\mathbb{T}}(x_1 \dots x_k)$, for any $x_1 \dots x_k$ in A^*. Then we obtain

$$\begin{aligned}
C_{\mathbb{S}}(x_1 \dots x_k x_{k+1}) &= I(C_{\mathbb{S}}(x_1 \dots x_k), x_{k+1}) \\
&= I(C_{\mathbb{T}}(x_1 \dots x_k), x_{k+1}) \\
&= I(J(C_{\mathbb{T}}(x_2 \dots x_k), x_1), x_{k+1}) \\
&= J(I(C_{\mathbb{T}}(x_2 \dots x_k), x_{k+1}), x_1) \\
&= J(I(C_{\mathbb{S}}(x_2 \dots x_k), x_{k+1}), x_1) \\
&= J(C_{\mathbb{S}}(x_2 \dots x_k x_{k+1}), x_1) \\
&= J(C_{\mathbb{T}}(x_2 \dots x_k x_{k+1}), x_1) \\
&= C_{\mathbb{T}}(x_1 x_2 \dots x_k x_{k+1}).
\end{aligned}$$

In particular, we have $C_{\mathbb{S}}(R(d)R(d')) = C_{\mathbb{T}}(R(d)R(d'))$, for any $d, d' \in \mathrm{D}_A$. Moreover, we have $C_{\mathbb{S}}(R(d)R(d')) \overset{(1)}{=} d \star_{\mathbb{S}} d'$ and $C_{\mathbb{T}}(R(d)R(d')) \overset{(1)}{=} d' \star_{\mathbb{T}} d$, for any $d, d' \in \mathrm{D}_A$. Then we deduce that $\mathbb{S}$ and $\mathbb{T}$ commute.

For any $d, d', d'' \in \mathrm{D}_A$, the equality $C_{\mathbb{S}}(R(d)R(d')R(d'')) = C_{\mathbb{T}}(R(d)R(d')R(d''))$ holds, and we have $C_{\mathbb{S}}(R(d)R(d')R(d'')) \overset{(1)}{=} (d \star_{\mathbb{S}} d') \star_{\mathbb{S}} d''$ and $C_{\mathbb{T}}(R(d)R(d')R(d'')) \overset{(1)}{=} (d'' \star_{\mathbb{T}} d') \star_{\mathbb{T}} d$. Then, the equality $(d \star_{\mathbb{S}} d') \star_{\mathbb{S}} d'' = (d'' \star_{\mathbb{T}} d') \star_{\mathbb{T}} d$ holds for any $d, d', d'' \in \mathrm{D}_A$. Since $\mathbb{S}$ and $\mathbb{T}$ commute, we obtain

$$(d'' \star_{\mathbb{T}} d') \star_{\mathbb{T}} d = d \star_{\mathbb{S}} (d'' \star_{\mathbb{T}} d') = d \star_{\mathbb{S}} (d' \star_{\mathbb{S}} d'').$$

Thus, we obtain $(d \star_{\mathbb{S}} d') \star_{\mathbb{S}} d'' = d \star_{\mathbb{S}} (d' \star_{\mathbb{S}} d'')$, for any d, d', d'' in D_A. Hence, the SDS $\mathbb{S}$ is associative. Similarly, one proves that if there is a right SDS opposite to a left SDS $\mathbb{S}$, then the SDS $\mathbb{S}$ is associative.

C Proof of Theorem 2

Consider a right associative SDS $\mathbb{S} = (\mathrm{D}_A, \ell_{\mathrm{l}}, I, R_{\mathbb{S}})$. One shows that for any d in D_A and $x_1 \dots x_p$ in A^* the following equality holds

$$I^*(d, \ell_{\mathrm{l}}(x_1 \dots x_p)) = I^*(I^*(d, x_1 \dots x_k), x_{k+1} \dots x_p). \tag{2}$$

To prove Theorem 2 it is sufficient to show that $\mathbb{S}$ is compatible with the equivalence relation $\sim_{\mathbb{S}}$ induced by $\mathcal{R}(A,\mathbb{S})$. Let us show that $R_{\mathbb{S}}C_{\mathbb{S}}(w) \sim_{\mathbb{S}} w$, for any $w \in A^*$. Every $w = x_1 \dots x_p$ in A^* can be written $w = R_{\mathbb{S}}(\iota_{\mathbb{S}}(x_1)) \dots R_{\mathbb{S}}(\iota_{\mathbb{S}}(x_p))$, where $\iota_{\mathbb{S}}$ denotes the inclusion map of A into $R_{\mathbb{S}}(\mathrm{D}_A)$. Since $\mathbb{S}$ is associative and $\mathcal{R}(A,\mathbb{S})$ is terminating, $\mathcal{R}(A,\mathbb{S})$ is convergent. Then the application of the rewriting rules of $\mathcal{R}(A,\mathbb{S})$ on w yield to the normal form $R_{\mathbb{S}}(\iota_{\mathbb{S}}(x_1)\star_{\mathbb{S}} \dots \star_{\mathbb{S}}\iota_{\mathbb{S}}(x_p))$ which is equal to $R_{\mathbb{S}}(C_{\mathbb{S}}(w))$. Hence, we obtain $R_{\mathbb{S}}C_{\mathbb{S}}(w) \sim_{\mathbb{S}} w$.

Suppose that for w and w' in A^* we have $w \sim_{\mathbb{S}} w'$. Let us show that, for any d in D_A, we have $I^*(d,w) = I^*(d,w')$. Note that for any w in A^* and d in D_A, the following equality holds

$$C_{\mathbb{S}}(R_{\mathbb{S}}(d)w) = I^*(\emptyset, \ell_1(R_{\mathbb{S}}(d)w)) = I^*(\emptyset, \ell_1(R_{\mathbb{S}}(d))\,\ell_1(w)) \overset{(2)}{=} I^*(I^*(\emptyset, \ell_1(R_{\mathbb{S}}(d)), \ell_1(w)) = I^*(d,w).$$

Since $w \sim_{\mathbb{S}} w'$, we have $R_{\mathbb{S}}(d)w \sim_{\mathbb{S}} R_{\mathbb{S}}(d)w'$. Then by the unique normal form property of the SRS $\mathcal{R}(A,\mathbb{S})$, we have $R_{\mathbb{S}}(C_{\mathbb{S}}(R_{\mathbb{S}}(d)w)) = R_{\mathbb{S}}(C_{\mathbb{S}}(R_{\mathbb{S}}(d)w'))$, for any d in D_A. Since $R_{\mathbb{S}}$ is injective, we obtain $C_{\mathbb{S}}(R_{\mathbb{S}}(d)w) = C_{\mathbb{S}}(R_{\mathbb{S}}(d)w')$. Hence, for any d in D_A, we have $I^*(d,w) = I^*(d,w')$. As a consequence, we obtain that the SDS $\mathbb{S}$ is compatible with the equivalence relation $\sim_{\mathbb{S}}$.

D Coherent presentations of monoids

We recall from [5] the notion of coherent presentation of monoids. Let R be an SRS on an alphabet X. For every rewriting rule β of R we will denote respectively by $s_1(\beta)$ and $t_1(\beta)$ the source and the target of β. We will denote by $R^\top$ the $(2,1)$-category freely generated by the SRS R, that is the free 2-category enriched in groupoid generated by the set of rules R, see [6]. The 2-cells of the $(2,1)$-category $R^\top$ corresponds to elements of the equivalence relation generated by R. A 2-*sphere* of $R^\top$ is a pair (f,g) of 2-cells in $R^\top$ such that $s_1(f) = s_1(g)$ and $t_1(f) = t_1(g)$.

An *extended presentation* of a monoid $\mathbf{M}$ is an SRS R presenting $\mathbf{M}$ extended by a globular extension Γ of the $(2,1)$-category $R^\top$, that is a set of *homotopy generators* $A : f \Rrightarrow g$ relating parallel 2-cells f and g in $R^\top$, respectively denoted by $s_2(A)$ and $t_2(A)$ and satisfying the globular relations $s_1s_2(A) = s_1t_2(A)$ and $t_1s_2(A) = t_1t_2(A)$. We will denote by $\Gamma^\top$ the free $(3,1)$-category generated by such an extended presentation. A *coherent presentation of a monoid* $\mathbf{M}$ is an extended presentation (R,Γ) of $\mathbf{M}$ such that the cellular extension Γ is a *homotopy basis* of the $(2,1)$-category $R^\top$, that is, for every 2-sphere γ of $R^\top$, there exists a homotopy generator in $R^\top$ with boundary γ.

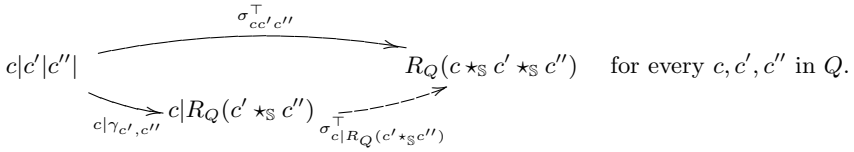
E Proof of Theorem 3

Let $\mathbb{S} = (\mathrm{D}_A, \ell, I, R_{\mathbb{S}})$ be an associative SDS and Q be a generating set of $\mathbb{S}$ such that the SRS $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ is terminating. Let σ be a normalization strategy of $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ that computes $C_{\mathbb{S}}$. Let us show that the SRS $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ is convergent. Any critical pair of $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ has the form $(\gamma_{c,c'}c'', c\gamma_{c',c''})$, for c, c', c'' in Q. Applying the normalisation σ, we have the following reduction diagram:

$$\begin{array}{ccc} & \xrightarrow{\gamma_{c,c'}c''} R_Q(c \star_{\mathbb{S}} c')|c'' & \xrightarrow{\sigma_{R_Q(c\star_{\mathbb{S}}c')|c''}} R_Q\big((c \star_{\mathbb{S}} c') \star_{\mathbb{S}} c''\big) \\ c|c'|c'' & & \\ & \xrightarrow{c\gamma_{c',c''}} c|R_Q(c' \star_{\mathbb{S}} c'') & \xrightarrow{\sigma_{c|R_Q(c'\star_{\mathbb{S}}c'')}} R_Q\big(c \star_{\mathbb{S}} (c' \star_{\mathbb{S}} c'')\big) \end{array}$$

which is confluent by the associativity of $\star_{\mathbb{S}}$. Hence, the SRS $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ is convergent by termination hypothesis. Moreover, by Lemma 1, the SRSs $\mathcal{R}(\mathbb{S})$ and $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ are Tietze-equivalent. Then the SRS $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ is a presentation of the structure monoid $\mathbf{M}(\mathbb{S})$ and thus the set $\mathrm{Nf}(Q, \mathbb{S})$ satisfies the cross-section property for $\mathbf{M}(\mathbb{S})$.

In particular, if the leftmost normalization strategy $\sigma^{\top}$ computes $C_{\mathbb{S}}$, then by [14] the SRS $\mathcal{R}(Q, \mathrm{D}_A, \mathbb{S})$ can be extended into a coherent convergent presentation by adjunction of



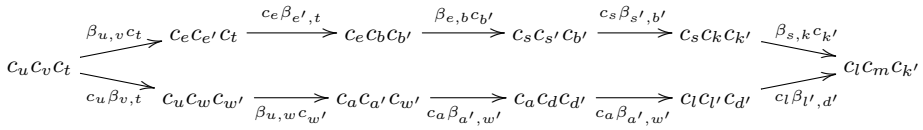
for every c, c', c'' in Q.

F Chinese coherent presentations

The *reduced presentation* of the Chinese monoid is the SRS whose set of generators is

$$Q_n = \{c_{yx} \mid 1 \leqslant x < y \leqslant n\} \cup \{c_{xx} \mid 1 < x < n\} \cup \{c_1, \ldots, c_n\},$$

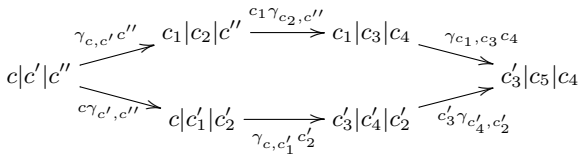
where $c_1, \ldots, c_n$ represent the initial generators $1, \ldots, n$, and whose rewriting rules are of the form $\gamma_{u,v} : c_u c_v \to c_w c_{w'}$, where $c_w c_{w'}$ is obtained by inserting c_v into c_u using the right insertion defined in [3]. We show that this presentation is a finite convergent presentation of the Chinese monoid and it can be extended into a coherent presentation by adjunction of



where the rewriting rules $\beta_{-,-}$ denote either a rewriting rule of the reduced presentation or an identity.

G Plactic coherent presentations

As an application of Theorem 3, consider the SDSs $\mathcal{Y}_n^{col}$ and $\mathcal{Y}_n^{row}$. By definition, the left-most normalisation strategy $\sigma^{\top}$ with respect to $\mathcal{R}(\mathrm{Col}_n, \mathrm{Yt}_n, \mathcal{Y}_n^{col})$ computes $C_{\mathcal{Y}_n^{col}}$. Then the SRS $\mathcal{R}(\mathrm{Col}_n, \mathrm{Yt}_n, \mathcal{Y}_n^{col})$ is extended into a finite coherent presentation by adjunction of, [7]:



where $c|c', c'|c'' \notin \mathrm{Yt}_n$, $R_{\mathrm{Col}_n}(c \star_{\mathcal{Y}_n^{col}} c') = c_1|c_2$, $R_{\mathrm{Col}_n}(c_2 \star_{\mathcal{Y}_n^{col}} c'') = c_3|c_4$, $R_{\mathrm{Col}_n}(c_1 \star_{\mathcal{Y}_n^{col}} c_3) = c_3'|c_5$, $R_{\mathrm{Col}_n}(c'' \star_{\mathcal{Y}_n^{row}} c') = c_1'|c_2'$, $R_{\mathrm{Col}_n}(c_1' \star_{\mathcal{Y}_n^{row}} c) = c_3'|c_4'$ and $R_{\mathrm{Col}_n}(c_2' \star_{\mathcal{Y}_n^{row}} c_4') = c_5|c_4$.

Coherence modulo relations

Benjamin Dupont and Philippe Malbos

Institut Camille Jordan, University of Lyon, France, {bdupont,malbos}@math.univ-lyon1.fr

Abstract – The computation of minimal convergent presentations for monoids, categories or higher-dimensional categories appear in low-dimensional combinatorial problems on these structures, such as coherence problems. A method to compute coherent presentations using convergent string rewriting systems was developed following works of Squier. In this approach, coherence results are formulated in terms of confluence diagrams of critical pairs. This work proposes an extension of these methods to string rewriting systems modulo.

1 Introduction

The computation of minimal convergent presentations for monoids, categories or higher-dimensional categories appear in low-dimensional combinatorial problems on these structures, such as coherence problems. A method to compute coherent presentations using convergent string rewriting systems was developed following works of Squier, see [9, 10]. In this approach, coherence results are formulated in terms of confluence diagrams of critical pairs. This work proposes an extension of these methods to algebraic or categorical structures having additional algebraic axioms, such as commutation, linearity or inverses. Using a notion of rewriting modulo, we show how to compute coherent presentations modulo algebraic axioms.

Rewriting modulo was developed in several approaches. The rules are split into two parts: oriented rules in a set R and non-oriented equations in a set E. The most naive approach of rewriting modulo is to use a rewriting system R/E consisting in rewriting on congruence classes modulo E, but this appears unefficient for analysis of confluence, see [1, Chapter 11]. Another approach of rewriting modulo has been considered by Huet in [11] where rewriting paths does not involve equivalence steps, and confluence is formulated modulo equivalence. Jouannaud and Kirchner enlarged this approach in [12] by providing completion methods for any rewriting system between R and R/E. Several other approaches have also been developed for term rewriting systems modulo to deal with various equational theories, see [2, 14, 17].

In this work, we extend Huet's approach to prove coherence results modulo algebraic relations, e.g. inverses for rewriting in groups, or commutation for linear rewriting. Indeed, in most cases, algebraic relations such as commutation cannot be oriented in a terminating way. Moreover, rewriting modulo can be used to delete some critical branchings that should not be considered in the analysis of coherence. This is the case for the computation of coherent presentations for algebraic structures such as groups or algebras.

Known approaches of rewriting for groups are mainly based on a presentation of groups as monoids with explicit inverses and explicit rules for inverse axioms. The SRS is thus defined on the set of generators of the group, their formal inverses, and the explicit rules for inverses, [3–6, 15]. However, coherent presentations of groups have to take into account that the presentation is modulo these inverse relations. The objective is to study confluence modulo the confluence diagrams induced by these relations, and to consider rewriting steps in the free group. This approach corresponds to rewriting on congruence classes modulo the equivalence given by the inverse relations, and thus is not suitable to study confluence. For this reason, we consider the weaker theory of rewriting modulo introduced by Huet. One of the main applications is to extend the coherent results obtained by rewriting methods on Artin-Tits monoids in [7] to Artin-Tits groups.

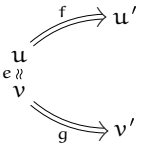
The second example of algebraic structure we consider is the axiomatic of vector spaces in the theory of linear rewriting developed in [8]. Actually, the critical branchings in linear rewriting are defined modulo the axioms of vector spaces, namely abelian groups and distributivity of the multiplicative law. For instance, if we denote by E the set of axiomatic rules of vector spaces and R is a linear rewriting system with two rules $3x \Rightarrow 2y$ and $2x \Rightarrow z$, then the branching $2y \Leftarrow 3x = 2x + x \Rightarrow z + x$ can be interpreted as a branching of R modulo E. In this way, the coherence result obtained on algebras in [8] can be formulated in terms of rewriting modulo.

This work presents a construction of coherent extensions of SRS modulo. In a first part, we recall the notion of confluence modulo as introduced by Huet, [11]. Then, we introduce the notion of coherent extension modulo, that corresponds to homotopy bases of SRS as defined in [10] when the set of axioms is empty. It is defined by a set of 3-cells modulo tiling all the spheres created by rewriting paths which are parallel modulo the axioms. In the last section, we enounce a generalization of Squier's coherence theorem to confluent SRS modulo. The proof of this result is given in Appendix A as well as some recalls on the categorical language on SRS used in this work in Appendix B.

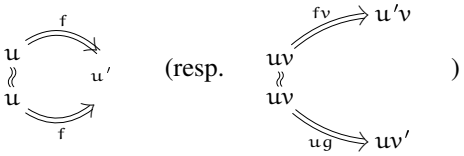
2 Rewriting modulo

Let us recall the notion of rewriting modulo a set of relations. In the sequel, all the SRS considered are defined on a same alphabet X. Given two SRS R and E, a rewiting with respect to R modulo relations defined by E consists in rewriting using rules of R on congruence classes modulo E. This corresponds to studying the rewriting system R/E defined by $u \Rightarrow_{R/E} v$ if and only if there exists strings u' and v' on X such that $u =_E u' \Rightarrow_R v' =_E v$. However, studying confluence of this rewriting system is complicated as explained in [1], so we use a weaker notion of confluence modulo as introduced by Huet in [11]. Whenever it exists, we denote by $\hat{u}$ a normal form of a string u on X with respect to R.

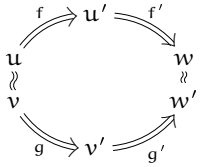
Equivalence modulo. Let consider the free $(2,1)$-category $E^\top$ generated by E (see Appendix B for categorical constructions). The 2-cells of $E^\top$ will be called *equivalences modulo* E. An equivalence modulo E of length equals to 1 is called a *one-step equivalence*. We denote by $\approx_E$ the equivalence relation generated by E. A *branching modulo* E of the SRS R is a pair (f,g) of 2-cells of the free 2-category R^* such that $s_1(f) \approx_E s_1(g)$ as depicted by the diagram on the side. We do not distinguish the branchings (f,g) and (g,f). Such a branching (f,g) is *local* if $\ell(f), \ell(g), \ell(e) \leqslant 1$ and $\ell(f) + \ell(g) + \ell(e) = 2$.



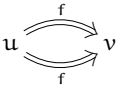
An *aspherical (resp. Peiffer) branching modulo* E of R is a pair (f,f) (resp. (fv, ug)) of 2-cells of R^* depicted by



A branching (f,g) is *confluent modulo* E if there exists 2-cells f' and g' in R^*, as in the diagram on the side. (f',g') is called a confluence modulo E. We will denote this by $u' \overset{E}{\vee} v'$. We say that R is *confluent modulo* E if all of its branchings are confluent modulo E.



Local branching. Local branchings belong to one of the following families:

i) *local aspherical* branchings, for a rewriting step f of R: 

ii) *local Peiffer* branchings: for rewriting steps f, g of R (resp. for a rewriting step f of R and a one-step equivalence e of E):

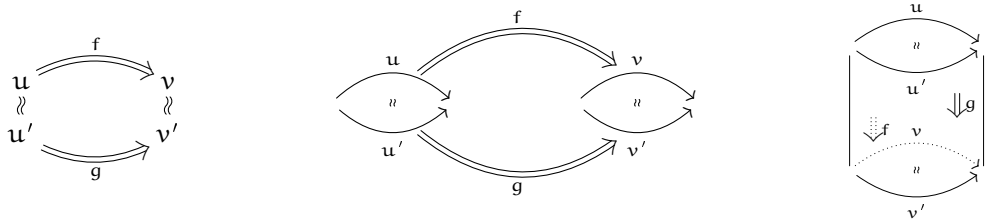
iii) *overlapping* branchings are the remaining local branchings, in which we distinguish two families:

A *critical branching modulo* E is an overlapping local branching that is minimal for the order generated by the relations $(f, g) \preccurlyeq (ufv, ugv)$.

Local Confluence modulo. A SRS R is *locally confluent modulo* a SRS E if any of its local branchings is confluent modulo E. Note that any aspherical and Peiffer branching being confluent modulo E, local confluence modulo E is equivalent to the confluence of overlappings modulo E. Huet show that under the assumption that the composite $\Rightarrow_R \cdot \approx_E$ is terminating, then R is confluent modulo E if and only if any overlapping branching of R modulo E is confluent modulo E, [11]. Under the same assumption, he shows that a SRS R is locally confluent modulo a SRS E if and only if any critical branching of R modulo E is confluent modulo E, [11].

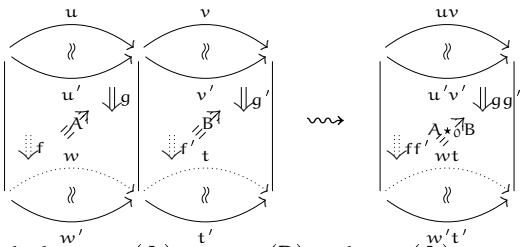
3 Coherent extensions modulo

Given two SRS R and E, a 2-*sphere modulo* E in the free $(2,1)$-category $R^\top$ is a pair (f, g) of 2-cells in $R^\top$ which are *parallel modulo* E, that is, $s_1(f) \approx_E s_1(g)$ and $t_1(f) \approx_E t_1(g)$ and such that f or g is not trivial. Note that the case f and g trivial produce a 2-sphere in $E^\top$. These 2-spheres do not fit in the construction of coherence extensions modulo E. A 2-sphere modulo E will be pictured by one of the following diagrams:

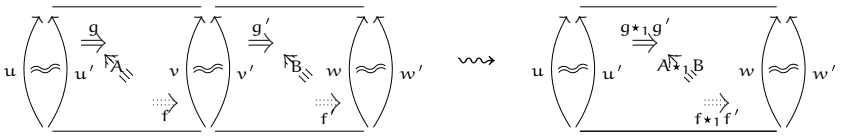


We will denote by $\mathrm{Sph}_E(R)$ the set of 2-spheres modulo E in $R^\top$. A *cellular extension of* $R^\top$ *modulo* E is a set Γ equipped with a map $\gamma : \Gamma \longrightarrow \mathrm{Sph}_E(R)$, whose elements are called 3-*cells modulo* E. We say that Γ is *coherent* if the map γ is surjective. A 3-cell A modulo E filling a 2-sphere (f, g) modulo E will be denoted by $A : f \Rrightarrow_E g$. We say that f (resp. g) is the 2-source (resp. 2-target) of A and we denote it by $s_2(A)$ (resp. $t_2(A)$). We define formal compositions $\star_0, \star_1, \star_2$ of 3-cells modulo E in a cellular extension Γ as pasting operations defined as follows:

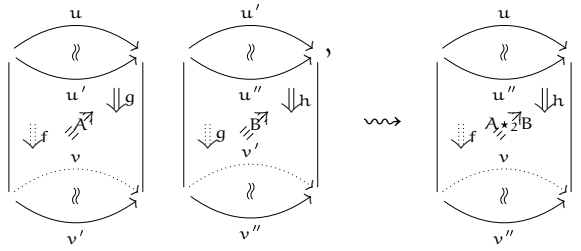
i) Given A and B in Γ, one defines $A \star_0 B$ as the 3-cell modulo E tiling the cylinder as follows:



ii) Given A and B in Γ such that $t_1 s_2(A) = s_1 s_2(B)$ and $t_1 t_2(A) = s_1 t_2(B)$, one defines a 3-cell modulo E denoted by $A \star_1 B$ tiling the following composite cylinder:



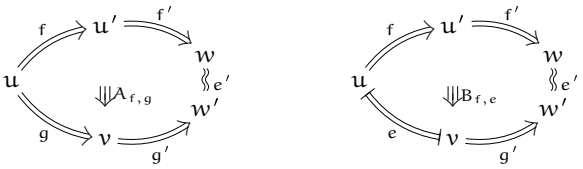
iii) Given A and B in Γ such that $t_2(A) = s_2(B)$, one defines a 3-cell modulo E denoted by $A \star_2 B$ tiling the cylinder obtained as follows:



Let Γ be a cellular extension of $R^\top$ modulo E. We will denote by $\mathcal{C}(\Gamma)$ the closure of Γ with respect to compositions $\star_0$, $\star_1$ and $\star_2$ of 3-cells of Γ and their formal inverses A^{-1} for $A \in \Gamma$ quotiented by the exchange relations $(A \star_i B) \star_j (A' \star_i B') = (A \star_j A') \star_i (B \star_j B')$ for any $0 \leqslant i < j \leqslant 2$ and the inverse relations $A \star_i A^- = 1_{s_i(A)}$ for any A in Γ and $i = 0, 1, 2$. When $\mathcal{C}(\Gamma)$ is a coherent extension of $R^\top$ modulo E, we say that Γ is *acyclic modulo* E.

4 Coherence from confluence modulo

Squier's completion. Suppose that R is a confluent SRS modulo a SRS E. A *Squier's completion modulo* E of R is a cellular extension modulo E of $R^\top$ whose elements are the 3-cells

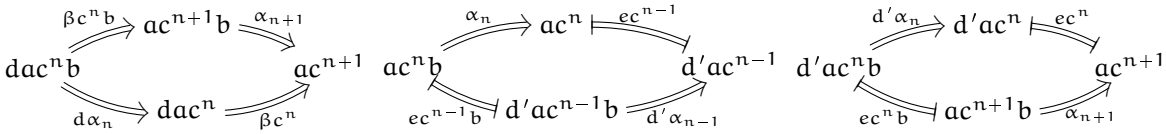


for any critical branching (f, g) and (f, e) of R modulo E, where f, g are rewriting steps of R and e is a one-step equivalence of E. Note that such a completion is not unique in general and depends on the rewriting sequences f', g' and the equivalence e' used to obtain the confluence diagrams.

4.1. Theorem (Coherence modulo). *Let* R *be a SRS confluent modulo a SRS* E *such that* $\Rightarrow_{\mathsf{R}} \cdot \approx_{\mathsf{E}}$ *is terminating, then any Squier's completion of* R *modulo* E *is acyclic.*

A proof of this result is given in Appendix A. As a consequence of Theorem 4.1, with the same hypothesis, one can prove that if Γ is an acyclic extension of $\mathsf{E}^{\top}$ then $\mathcal{C}(\mathcal{S}(\mathsf{R},\mathsf{E})) \sqcup \Gamma$ is a coherent extension of $(\mathsf{R} \sqcup \mathsf{E})^{\top}$. In particular when E is convergent, we fix a Squier completion $\mathcal{S}(\mathsf{E})$ and we get that $\mathcal{C}(\mathcal{S}(\mathsf{R},\mathsf{E})) \sqcup \mathcal{S}(\mathsf{E})$ is a coherent extension of $(\mathsf{R} \sqcup \mathsf{E})^{\top}$.

Example. Let R be the SRS on $X = \{a, b, c, d, d'\}$ defined by the rules $ab \overset{\alpha_0}{\Rightarrow} a$ and $da \overset{\beta}{\Rightarrow} ac$. We complete the SRS R into a confluent SRS by adding the rules $ac^n b \overset{\alpha_n}{\Rightarrow} ac^n$ for all n in $\mathbb{N}$ in R. Let us consider the SRS E defined on X with the rule $d'a \overset{e}{\Rightarrow} ac$. A Squier's completion of R modulo E is then given by a family of 3-cells modulo E tiling the following confluence modulo diagrams:



Note that up to a diagram rotation, the last two families of confluence diagrams are the same, so the coherent extension of R modulo E consists in the two families of 3-cells given by the first two confluence modulo diagrams. We recover the coherent extension of the SRS $\mathsf{R} \sqcup \mathsf{E}$ given in [13].

Finiteness conditions modulo. In the case where E is empty, Theorem 4.1 is the Squier's theorem for SRS, [16], see [10] for a polygraphic proof. From Squier's result, it follows the homotopical finiteness from convergence: if a monoid admits a presentation by a finite convergent SRS, then it has finite derivation type (FDT). From Theorem 4.1, we deduce a new finiteness condition modulo. If a monoid admits a presentation by a finite convergent SRS R modulo E, then it has *FDT modulo* E, that is, R admits a finite cellular extension Γ modulo E such that $\mathcal{C}(\Gamma)$ is acyclic. If the SRS R modulo E has FDT modulo E and E has FDT, then the SRS $\mathsf{R} \sqcup \mathsf{E}$ has FDT. In particular, if R is a finite confluent SRS modulo a finite convergent SRS such that $\Rightarrow_{\mathsf{R}} \cdot \approx_{\mathsf{E}}$ terminating, then $\mathsf{R} \sqcup \mathsf{E}$ has FDT.

5 Conclusion and work in progress

In this work, we have presented a coherence result for SRS modulo a set of axioms. This result is based on the notion of confluence modulo introduced by Huet. However, completion procedures for such SRS are missing. We expect that some completion methods given in [2, 12, 17] could be adapted to compute Squier's completion of non confluent SRS modulo. In particular, with the axioms of group, naive completion modulo induces infinitely many completion steps due to overlapping branchings between a rule and an equivalence obtained by adding elements of the form xx^{-1}. The objective is to define an appropriate completion procedure allowing to avoid this infinite completion. One approach is to consider a restriction of local obstruction of confluence by considering rewriting step conditioned by algebraic context.

REFERENCES

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

[2] Leo Bachmair and Nachum Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, 67(2):173 – 201, 1989.

[3] Fabienne Chouraqui. Rewriting systems and embedding of monoids in groups. *Groups Complex. Cryptol.*, 1(1):131–140, 2009.

[4] Fabienne Chouraqui. The Knuth-Bendix algorithm and the conjugacy problem in monoids. *Semigroup Forum*, 82(1):181–196, 2011.

[5] Volker Diekert, Andrew Duncan, and Alexei G. Myasnikov. Cyclic rewriting and conjugacy problems. *Groups Complex. Cryptol.*, 4(2):321–355, 2012.

[6] Volker Diekert, Andrew J. Duncan, and Alexei G. Myasnikov. Geodesic rewriting systems and pregroups. In *Combinatorial and geometric group theory*, Trends Math., pages 55–91. Birkhäuser/Springer Basel AG, Basel, 2010.

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

[8] Yves Guiraud, Eric Hoffbeck, and Philippe Malbos. Convergent presentations and polygraphic resolutions of associative algebras. arXiv:1406.0815v2, December 2017.

[9] Yves Guiraud and Philippe Malbos. Coherence in monoidal track categories. *Math. Structures Comput. Sci.*, 22(6):931–969, 2012.

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

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

[12] Jean-Pierre Jouannaud and Helene Kirchner. Completion of a set of rules modulo a set of equations. In *Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '84, pages 83–92, New York, NY, USA, 1984. ACM.

[13] Yves Lafont. A new finiteness condition for monoids presented by complete rewriting systems (after Craig C. Squier). *J. Pure Appl. Algebra*, 98(3):229–244, 1995.

[14] Claude Marché. Normalized rewriting: an alternative to rewriting modulo a set of equations. *J. Symbolic Comput.*, 21(3):253–288, 1996.

[15] John Pedersen and Margaret Yoder. Term rewriting for the conjugacy problem and the braid groups. *J. Symbolic Comput.*, 18(6):563–572, 1994.

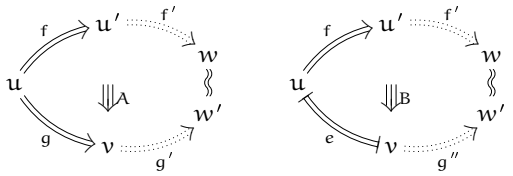
[16] Craig C. Squier, Friedrich Otto, and Yuji Kobayashi. A finiteness condition for rewriting systems. *Theoret. Comput. Sci.*, 131(2):271–294, 1994.

[17] Patrick Viry. Rewriting modulo a rewrite system. Technical report, 1995.

A. Proof of Theorem 4.1

Proof. Let us consider a Squier's completion $\mathcal{S}(R, E)$ of R modulo E.
Step 1. Prove that, for every local branching (f, g) and (f, e) of R modulo E with f, g in R and e in E,

there exist 2-cells f' and g' in R^* and 3-cells $A : f \star_1 f' \Rightarrow g \star_1 g'$ and $B : f \star_1 f' \Rightarrow e \star_1 g'$ modulo E in $\mathcal{C}(\mathcal{S}(R, E))$, as in the following diagram:



In the case of a local aspherical branching, we set A as an identity. For a local Peiffer branching (f, g) with f, g in R, we can choose f' and g' such that $f \star_1 f' = g \star_1 g'$ and we set A an identity. For a local Peiffer branching (f, e) with f in R and e in E, we can choose f' as the empty 2-cell, $g'' = f$ and the right equivalence being e so that B is also an identity. Moreover, if we have an overlapping branching (f, g) (resp. (f, e)) that is not critical, we have $(f, g) = (uhv, ukv)$ (resp. $(f, e) = (uhv, ue'v)$) for some u, v in X^* such that both (h, k) and (h, e') are critical. We consider the 3-cells $A' : f \star_1 f' \Rrightarrow_E g \star_1 g'$ and $B' : f \star_1 f' \Rrightarrow_E e \star_1 g''$ corresponding respectively to the critical branchings (h, k) and (h, e'). We conclude by setting $f' = uh'v \quad g' = uk'v \quad g'' = ue'v \quad A' = u \star_0 A' \star_0 v \quad B = u \star_0 B' \star_0 v$.

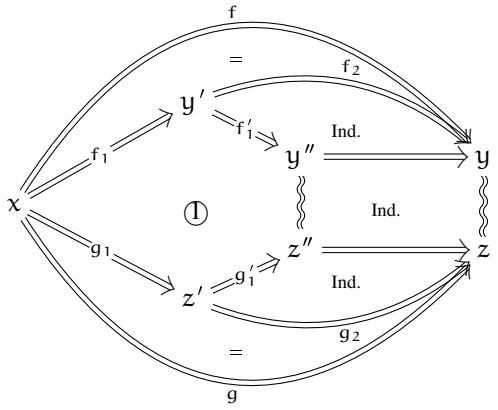
Step 2. We prove that, for any 2-cells $f : x \Rightarrow y$ and $g : x \Rightarrow z$ of R^*, there exists a 3-cell modulo E from f to g in $\mathcal{C}(\mathcal{S}(R, E))$. To do this, we decompose the 2-cells f and g into $f = f_1 \star_1 f_2$ and $g = g_1 \star_1 g_2$ where $f_1 : x \Rightarrow y'$ and $g_1 : x \Rightarrow z'$ are rewriting steps of R, and f_2,g_2 are in R^*. Then (f_1, g_1) is a local branching of R modulo E and we use local confluence modulo E to get 1-cells y'' and z'' in X^* and 2-cells $f_1' : y' \Rightarrow y''$ and $g_1' : z' \Rightarrow z''$ in R^* with $y'' \approx_E z''$. Using Step 1, we get a 3-cell $A : f_1 \star_1 f_1' \Rrightarrow_E g_1 \star_1 g'1$. We construct a 3-cell modulo E from f to g using Noetherian induction principle from [11] defined as follows: we fix an auxiliary string rewriting system R^{aux} with only one 0-cell, whose 1-cells are the pairs (x, y) of elements of X^*. R^{aux} contains a 2-cell $(x, y) \Rightarrow_{R^{aux}} (x', y')$ in any of the following situation:

- there exist 2-cells $x \Rightarrow x'$ and ($y \Rightarrow y'$ or $x \Rightarrow y'$) in R^*;
- there exists a 2-cell $y \Rightarrow y'$ in R and an equivalence $x \approx_E x'$ in E;
- there exist 2-cells $y \Rightarrow x'$ and $y \Rightarrow y'$ in R^*;
- $x \overset{e}{\approx}_E y \approx_E x' \overset{e'}{\approx}_E y'$ and $\ell(e) < \ell(e')$.

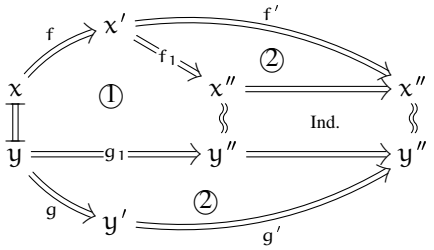
Following [11, Proposition 2.2], if $\Rightarrow_R \cdot \approx_E$ is terminating, then so is R^{aux}. Let us apply Noetherian induction on R^{aux} with the following property:

$$\mathcal{P}(x, y) : \quad x \approx_E y \;\Rightarrow\; \forall\, x', y' \mid x \overset{*}{\Rightarrow}_P x' \,\&\, y \overset{*}{\Rightarrow}_R \Rightarrow x' \overset{E}{\vee} y'$$

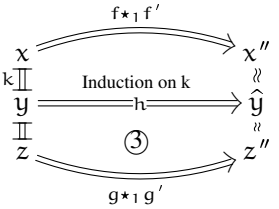
This leads to the diagram on the right which enables to construct a 3-cell $A : f \Rightarrow_E g$ in $\mathcal{C}(\mathcal{S}(R, E))$.



Step 3. We now prove that for each rewriting steps $f : x \Rightarrow x'$ and $g : y \Rightarrow y'$ in R such that $x \overset{e}{\approx}_E y$, there exist 2-cells $f' : x' \Rightarrow x''$, $g' : y' \Rightarrow y''$ in R^* and a 3-cell modulo E from $f \star_1 f'$ to $g \star_1 g'$. We will prove the result by induction on $\ell(e)$. If $\ell(e) = 0$, this is Step 1. Suppose that $\ell(e) = 1$, that is $x \models\!\!= y$. By local confluence of R modulo E, looking at the local branching (f, e), we get the existence of 2-cells $f_1 : x' \Rightarrow x''$, $g_1 : y \Rightarrow y''$ in R^* with $x'' \approx_E y''$. By Step 1, there exists a 3-cell modulo E in $\mathcal{C}(\mathcal{S}(R, E))$ from $f \star_1 f_1$ to g_1. We construct the 3-modulo E from $f \star_1 f'$ to $g \star_1 g'$ using Noetherian's induction as illustrated by the following diagram.

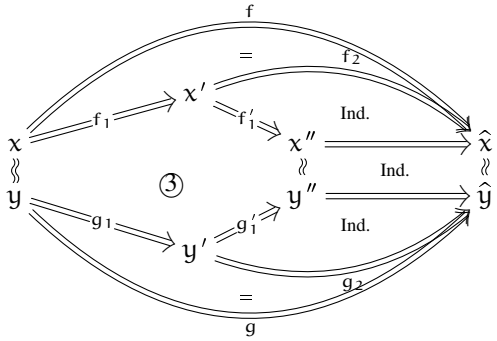


Hence the result is proved for $\ell(e) = 1$. Suppose the result proved for $\ell(e) = k > 1$ and let us prove the result for $\ell(e) = k + 1$. Suppose that $x \overset{k+1}{\models\!\!=} y$, we decompose this reduction by $x \overset{k}{\models\!\!=} z \models\!\!= y$. We fix a 2-cell $h : y \Rightarrow \hat{y}$ in R^*. By confluence modulo E, there exists 2-cells $f' : x' \Rightarrow x''$ and $g' : z' \Rightarrow z''$ in R^* such that $x'' \approx_E \hat{y} \approx_E z''$. We construct a 3-cell modulo E between $f \star_1 f'$ and $g \star_1 g'$ as depicted on the diagram on the right.

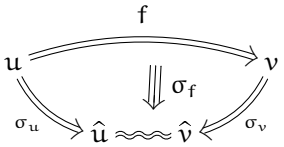


Step 4. Now, let us prove that for any 2-cells $f : x \Rightarrow \hat{x}$ and $g : y \Rightarrow \hat{y}$ with $x \overset{e}{\approx}_E y$, there exists a 3-cell $A : f \Rightarrow_E g$ modulo E in $\mathcal{C}(\mathcal{S}(R, E))$. Let us first write $f = f_1 \star_1 f_2$ and $g = g_1 \star_1 g_2$ where $f_1 : x \Rightarrow x'$ and $g_1 : y \Rightarrow y'$ are rewriting steps in R and $f_2 : x' \Rightarrow \hat{x}$, $g_2 : y' \Rightarrow \hat{y}$ are 2-cells in R^*. Using the confluence modulo E on the triple (f_1, e, g_1), we get the existence of 1-cells x'', y'' and 2-cells $f_1' : x' \Rightarrow x''$ and $g_1' : y' \Rightarrow y''$ such that $x'' \approx_E y''$. According to Step 3, there exists a 3-cell modulo E in $\mathcal{C}(\mathcal{S}(R, E))$ from $f_1 \star_1 f_1'$ to $g_1 \star_1 g_1'$. By Noetherian induction principle, we get the

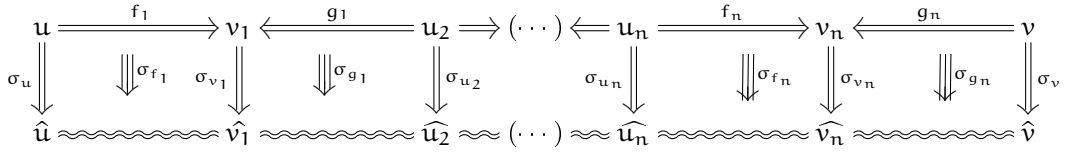
following diagram allowing us to construct the 3-cell A:



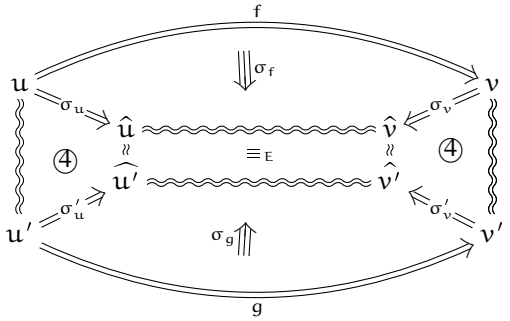
Step 5. We prove that every 2-sphere modulo E of $R^{\top}$ is the boundary of a 3-cell modulo E of $\mathcal{C}(\mathcal{S}(R,E))$. First, let us consider a 2-cell $f : u \Rightarrow v$ in R^*. Using confluence modulo E of R, there exist 2-cells in R^* $\sigma_u : u \Rightarrow \hat{u}$ and $\sigma_v : v \Rightarrow \hat{v}$ in R^* such that $\hat{u} \approx_E \hat{v}$. By construction, the 2-cells $f \star_1 \sigma_v$ and σ_u are parallel modulo E and their respective targets are normal forms. By Step 4, there exists a 3-cell modulo E in $\mathcal{C}(\mathcal{S}(R,E))$ from $f \star_1 \sigma_v$ to σ_u as in the diagram on the right.



Now, let us consider a 2-cell $f : u \Rightarrow v$ of $R^{\top}$. By construction of $R^{\top}$, the 2-cell f can be decomposed in a non unique way into a zigzag sequence $f_1 \star g_1^{-1} \star_1 \cdots \star_1 f_n \star_1 g_n^{-1}$ with source u and target v where each f_i and g_i is a 2-cell of R^*. We define a 3-cell modulo $\sigma_f : f \star_1 \sigma_v \Rrightarrow_E \sigma_u$ in $\mathcal{C}(\mathcal{S}(R,E))$ as the following composition:



Proceeding similarly for any other 2-cell $g : u' \Rightarrow v'$ of $R^{\top}$, we get a 3-cell $\sigma_g : g \star_1 \sigma_{v'} \Rrightarrow_E \sigma_u$ in $\mathcal{C}(\mathcal{S}(R,E))$. In this way, for any 2-sphere (f,g) modulo E in $R^{\top}$, there exists a 3-cell modulo $f \Rrightarrow_E g$ in $\mathcal{C}(\mathcal{S}(R,E))$ given by the following composition:



□

B. Categorical formulation of string rewriting systems

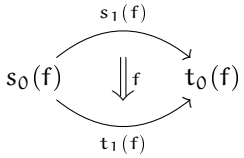
In this work, the constructions on SRS are formulated in categorical language. In this part, we recall the notions used in the text for a reader unfamiliar to this language. We refer to [10] for a deeper presentation of categorical formulation of SRS.

1-categories of strings. Given an alphabet X, we denote by X^* the free monoid generated by X. This monoid can be seen as the free 1-category generated by X, that is a 1-category with only one 0-cell and whose 1-cells are strings made of elements of X. Having only one 0-cell, any two 1-cells of X^* are composable and the composition corresponds to concatenation of strings. This concatenation is associative and unitary with the empty string as unit.

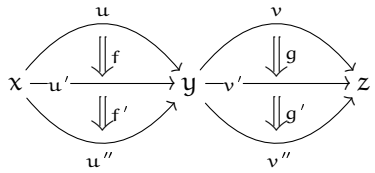
2-categories of rewriting steps. Recall that a 2-category $\mathcal{C}$ is defined by a set $\mathcal{C}_0$ of 0-cells, a set $\mathcal{C}_1$ of 1-cells and a set $\mathcal{C}_2$ of 2-cells and equipped with two compositions $\star_0$ for 1-cells and 2-cells and $\star_1$ for 2-cells. A 2-category is equipped with source and target maps making it a 2-graph, that is a digram in the category of sets:

$$\mathcal{C}_0 \underset{t_0}{\overset{s_0}{\leftleftarrows}} \mathcal{C}_1 \underset{t_1}{\overset{s_1}{\leftleftarrows}} \mathcal{C}_2$$

where the maps satisfy the globular relations: $s_0 s_1 = s_0 t_1$ and $t_0 s_1 = t_0 t_1$. For any $1 \leqslant i < j \leqslant 2$, the i-cell $s_i(f)$ (resp. $t_i(f)$) is called the i-*source* (resp. i-*target*) of a j-cell f. A 2-cell f in $\mathcal{C}$ can be pictured by

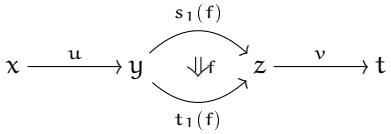


The composition $\star_0$ and $\star_1$ are associative and unitary and compatible with source and target maps. They also satisfy the exchange law, that is, for any situation



the equality $(f \star_0 g) \star_1 (f' \star_0 g') = (f \star_1 f') \star_0 (g \star_1 g')$ holds.

Given a SRS R on an alphabet X, one can construct the free 2-category generated by R, denoted by R^* and defined as follows. It has only one 0-cell, its 1-cells are strings on X and its 2-cells are rewriting paths of R. The $\star_0$-composition in R^* corresponds to concatenation of strings, and the $\star_1$-composition is the sequential composition of rewritings of R. Each 2-cell f of R^* can be decomposed into a sequence $f = f_1 \star_1 f_2 \star_1 \ldots \star_1 f_k$, where each f_i is a 2-cell corresponding to a rewriting step of the form:



that we will denote by $u f v$. The length of a 2-cell f in R^*, denoted by $\ell(f)$ is the minimal number of rewriting steps in any $\star_1$-decomposition of f. We denote by $u \Rightarrow_R v$ if there exists a 2-cell in R^* of length 1, that is u rewrites to v in one R-step.

$(2,1)$-categories of equivalence. Let E be a SRS on an alphabet X. The free $(2,1)$-category generated by E, denoted by $E^\top$, is the free 2-category on E in which all the 2-cells are invertible with respect to the $\star_1$-composition. That is its 0-cells, 1-cells and 2-cells are those of E^*, and any 2-cell f of $E^\top$ has an inverse $f^- : t_1(f) \Rightarrow s_1(f)$ with respect the $\star_1$-composition satisfying the relations $f \star_1 f^- = 1_{s_1(f)}$ and $f^- \star_1 f = 1_{t_1(f)}$. The 2-cells of the $(2,1)$-category $E^\top$ corresponds to elements of the equivalence relation generated by E, that we will denote by $\approx_E$.

The diamond lemma for free modules

Cyrille Chenavier

Université Paris-Est Marne-la-Vallée

Abstract

We study rewriting systems over free modules, that is linear combinations of free generators with noninvertible coefficients. We provide a sufficient condition in terms of local confluence restricted to generators for the global rewrite relation to be confluent: this condition is formulated in terms of syzygies. When the coefficients belong to a domain, we equip the set of syzygies with a module structure, which provides a finer criterion: the local confluence has to be checked over a subset of syzygies, namely a generating set for the module structure.

1 Introduction

The diamond lemma for noncommutative polynomials was introduced by Bergman [4] for computing normal forms in noncommutative algebras using rewriting theory. The diamond lemma together with the works of Bokut [5] gave birth the theory of noncommutative Gröbner bases [7]. The latter provides applications in various areas of noncommutative algebra: study of embedding problems, this was the motivation of Bokut and Bergman, homological algebra [1, 6] or Koszul duality [2, 3], for instance.

The diamond lemma is based on the observation that the set of noncommutative polynomials admitting a unique normal form is a vector space. Hence, the set of noncommutative monomials being a linear basis of noncommutative polynomials, it is sufficient to check the local confluence property over these monomials. The diamond lemma asserts that when the so called overlapping/inclusion ambiguities (which correspond to critical pairs for term rewriting) are joinable, then every monomial admits a unique normal form, so that the global rewrite relation is confluent.

In this work, we are interested in the study of linear combinations of monomials where the coefficients in these combinations do not form a field. In this framework, elements with a unique normal form do not form a subspace anymore: consider for instance a rewrite rule $2y \longrightarrow x$, where the coefficients belong to the ring of integers $\mathbb{Z}$. Since 2 is not inverible in $\mathbb{Z}$, the monomial y is a normal form but $y + y = 2y$ is not a normal form! This observation has the following consequence: a rewrite relation such that every monomial admits a unique normal form has no reason to be confluent. For instance, consider the two rewrite rules $2y \longrightarrow -x$ and $2x \longrightarrow -y$. Then, one can show that for every $n \in \mathbb{Z}$, nx and ny admit a unique normal form, but $2x + 2y$ rewrites both in x and y which are not joinable!

In Theorem 4.5, we present an analogous version of the diamond lemma for rewriting systems over linear combinations with noninveritble coefficients. This work does not concern noncommutative polynomials but the more general case of *free left module* (formal definitions are given at the beginning of the next section): we do not take into account the structure of monomials. The adaptation of the criterion of Theorem 4.5 to noncommutative polynomials with noninvertible coefficients is a further work. Two other further works should be mentionned there: when the coefficients are $\mathbb{Z}$, the underlying module structure is the one of abelian groups, so that we wish to develop rewriting theory in this context. Another perspective is the study of the case where monomials are terms of the λ-calculus, which is the framework of *algebraic λ-calculus* [8].

2 Rewrite systems over free left modules

Throughout the paper, we fix a not necessarily commutative ring $\mathbf{R}$ and a set X. We denote by $\mathbf{R}X$ the free left module over X, that is the set of finite formal linear combinations of elements of X with coefficients in $\mathbf{R}$. Given such two elements $f = \sum r_x x$ and $g = \sum s_x x$, the sum $f+g$ is equal to $\sum (r_x + s_x)\, x$ and the left product of $r \in \mathbf{R}$ with f is equal to $\sum (r r_x)\, x$, where $r r_x$ is the product of r and r_x in $\mathbf{R}$.

A set $\mathscr{R}$ of rewrite rules over $\mathbf{R}X$ is said to be *left-monomial* if its elements are of the form $rx \longrightarrow f$, where r, x and f belong to $\mathbf{R}$, X and $\mathbf{R}X$, respectively. Our first objective is to extend $\mathscr{R}$ into a rewrite relation on $\mathbf{R}X$, still written $\longrightarrow$, in such a way that the congruence relation induced by $\longrightarrow$ is the left ideal generated by $\mathscr{R}$. In other words, we want to have the following equivalence:

$$f \overset{*}{\longleftrightarrow} g \iff f - g = \sum s\,(rx - f)\,, \tag{1}$$

with the sum over a finite set of indexes $(s,\ rx \longrightarrow f) \in \mathbf{R} \times \mathscr{R}$. For that, we choose representatives for every left class modulo r, so that every element $s \in \mathbf{R}$ admits a decomposition $r_1 r + r_2$ where r_2 is the chosen representative of the left class of s. The rewrite relation induced by $\mathscr{R}$ is defined by

$$(r_1 r + r_2)\, x \ + \ g \ \longrightarrow \ r_1 f \ + \ r_2 x \ + \ g, \tag{2}$$

where x belongs with a zero coefficient in the decomposition of g. The rewrite relation (2) satisfies the equivalence (1).

Example 2.1. When the ring $\mathbf{R}$ is left euclidean, we choose the representatives of left classes as the set of remainders for the euclidean division. Here, we present the explicit description of the rewrite relation for two examples of euclidean rings: the ring of integers $\mathbb{Z}$ and a commutative field $\mathbb{K}$. Consider a rewrite rule $nx \longrightarrow f$ over $\mathbb{Z}X$ and an integer m. By euclidean division, m is equal to $qn + r$. Then, $mx + g$ rewrites into $qf + rx + g$. A commutative field $\mathbb{K}$ is an euclidean ring where the euclidean division of μ by λ is $\mu = (\mu/\lambda)\,\lambda$. Then, the rewrite rule $\lambda x \longrightarrow f$ induces the rewrite step $\mu x + g \longrightarrow (\mu/\lambda)\, f + g$.

3 Compatible termination order

In the next section, we formulate the diamond lemma for rewrite relations over $\mathbf{R}X$ induced by a left-monomial set of rewrite rules $\mathscr{R}$. For that, we assume that the rewrite relation induced by $\mathscr{R}$ satisfies the following hypothesis:

$$\forall\ (rx \longrightarrow f,\ h) \in \mathscr{R} \times \mathbf{R}X,\ rx + h \downarrow f + h, \tag{3}$$

where $f \downarrow g$ means that f and g are joinable. Moreover, we also need that the rewrite relation is equipped with a *compatible termination order* defined in Definition 3.1. In this definition we use the following notation: given $f \in \mathbf{R}X$, we denote by $\operatorname{supp}(f)$ the set of elements of X which belong to the decomposition of f with nonzero coefficient.

Definition 3.1. A *termination order compatible* with $\mathscr{R}$ is a is a well-founded order $\preceq$ over $\mathbf{R}X$ such that for every $f,\ g,\ h \in \mathbf{R}X$ and every $a,\ b \in \mathbf{R}$ the following conditions are satisfied:

i. if $f \longrightarrow g$, then $g \preceq f$,

ii. if $g \preceq f$ and $\mathrm{supp}(h) \cap \mathrm{supp}(f) = \emptyset$, then $g + h \preceq f + h$,

iii. if $f \preceq ax$, $g \preceq by$ and $ax + by \neq 0$, then $f + g \preceq ax + by$,

iv. if $f \preceq ax$ and $ab \neq 0$, then $bf \preceq (ba)x$.

Example 3.2. Assume that that for every $rx \longrightarrow f \in \mathscr{R}$, x does not belong to $\mathrm{supp}(f)$. Then, one can show that $\mathscr{R}$ satisfies (3). Moreover, assume that X is equipped with a well-founded order $\preceq$. Then, we define the order on $\mathbf{R}X$, still written $\preceq$, as the restriction of the multi-set order to finite subsets of X: we have $g \prec h$ if $\mathrm{supp}(g) \cap \mathrm{supp}(h) \neq \emptyset$ and for every $x \in \mathrm{supp}(g)$ such that $x \notin \mathrm{supp}(h)$, there exists $y \in \mathrm{supp}(h)$ such that $y \notin \mathrm{supp}(g)$ and $x \prec y$. Then, we can show that $\preceq$ is compatible with $\mathscr{R}$.

The diamond lemma presented in the next section concerns rewrite systems satisfying the hypothesis (3) and equipped with a compatible termination order. In the sketch of proof of the diamond lemma, we we use Lemma 3.3. We need the following definition: given $f \in \mathbf{R}X$, we say that the rewrite relation $\longrightarrow$ is *locally confluent at* f if for every $g, h, k \in \mathbf{R}X$ such that $g \prec f, h \prec f, k \prec f, g \longrightarrow h$ and $g \longrightarrow k$, we have $h \downarrow k$.

Lemma 3.3. *Assume that $\mathscr{R}$ is equipped with a compatible termination order and satisfies the hypothesis (3) and that $\longrightarrow$ is locally confluent at f. For every $f_1, f_2, g_1, g_2 \preceq f$ such that $f_1 \downarrow g_1$ and $f_2 \downarrow g_2$, and for every $r \in \mathbf{R}$, we have $f_1 + f_2 \downarrow g_1 + g_2$ and $rf_1 \downarrow rg_1$.*

4 The diamond lemma

The diamond lemma [4] gives a criterion for testing local confluence over so called *critical pairs*. In Corollary 4.5, we formulate the diamond lemma for rewriting systems over free modules, which consists in testing local confluence for generating sets of *syzygies*. These generating sets are analogous to critical pairs in our framework.

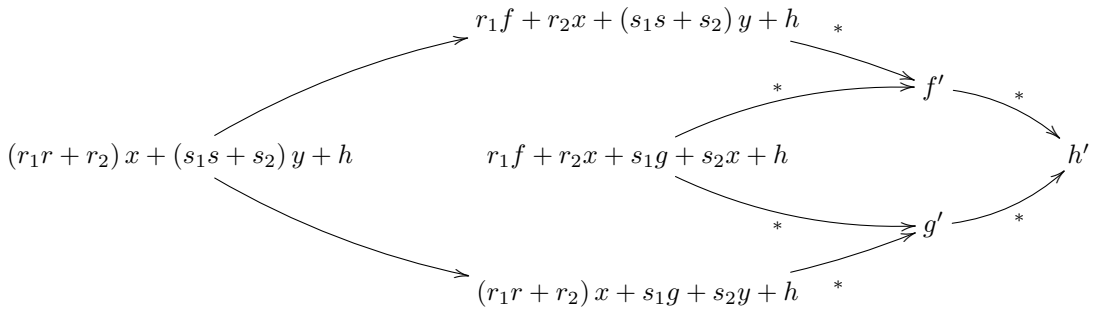
Definition 4.1. Let $p = (rx \longrightarrow f, sx \longrightarrow g)$ be a pair rewrite rules whose left-hand side are multiple of a common element x. A *syzygy* of p is a tuple (r_1, r_2, s_1, s_2) of elements of $\mathbf{R}$ such that r_2 and s_2 are the chosen representatives of their left classes modulo r ans s, respectively, and $r_1 r + r_2 = s_1 s + s_2$. The set of syzygies of p is written $\mathbf{syz}(p)$. Moreover, a syzygy (r_1, r_2, s_1, s_2) is said to be *confluent* if we have $r_1 f + r_2 x \downarrow s_1 g + s_2 x$.

Theorem 4.2. *Let $\mathscr{R}$ be a left-monomial set of rewrite rules satisfying hypothesis (3) and let $\preceq$ be a termination order compatible with $\longrightarrow$. The rewrite relation $\longrightarrow$ is confluent if and only if for every pair of rewrite rules $p = (rx \longrightarrow f, sx \longrightarrow g)$, every syzygy of p is confluent.*

Sketch of proof. Let $(r_1, r_2, s_1, s_2) \in \mathbf{syz}(p)$. Letting $h = (r_1 r + r_2)x = (s_1 s + s_2)x$, we observe that h rewrites into $r_1 f + r_2 x$ and $s_1 g + s_2 x$, which shows the direct implication.

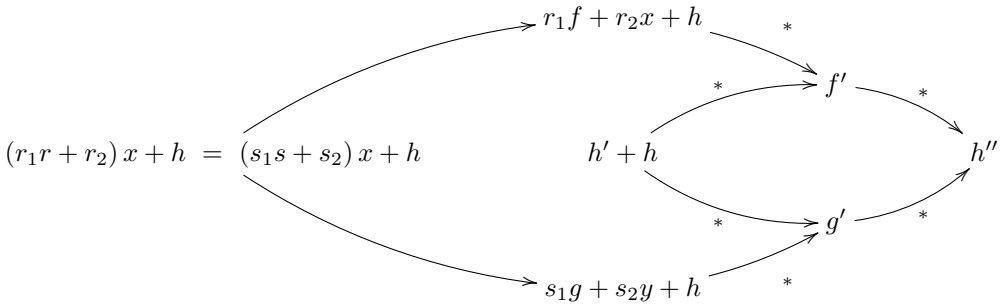
Assume that for every $p = (rx \longrightarrow f, sx \longrightarrow g)$ and for every $(r_1, r_2, s_1, s_2) \in \mathbf{syz}(p)$, we have $r_1 f + r_2 x \downarrow s_1 g + s_2 x$ and let us show that $\longrightarrow$ is confluent. The rewrite relation $\longrightarrow$ is terminating by definition of compatibility with a termination order, so that it is sufficient to show that it is locally confluent, or equivalently that is locally confluent at u for every u. We show the latter by induction on u: assume that $\longrightarrow$ is confluent at every $v \preceq u$ and that two rewrite rules $rx \longrightarrow f$ and $sy \longrightarrow g$ apply to u. Two cases have to be investigated according to $x \neq y$ or $x = y$ for proving that these two rewrite rules provide joinable terms.

First, if $x \neq y$, we let $u = (r_1 r + r_2)\, x + (s_1 s + s_2)\, y + h$ and we have the following confluence diagram:



The term f' (respectively g') and the two arrows coming to f' (respectively g') exist by hypothesis (3). By definition of a compatible rewrite order, we have $r_1 f + r_2 x + s_1 g + s_2 x + h \preceq (r_1 r + r_2)\, x + (s_1 s + s_2)\, y + h$, so that $\longrightarrow$ is confluent at $r_1 f + r_2 x + s_1 g + s_2 x + h$ by induction hypothesis, which gives h' and the two arrows coming to h'.

If $x = y$, we let $u = (r_1 r + r_2)\, x + h = (s_1 s + s_2)\, x + h$ and we have the following confluence diagram:



The tuple $(r_1,\ r_2,\ s_1,\ s_2)$ is a syzygy, so that there exists h' such that $r_1 f + r_2 x \xrightarrow{*} h' \xleftarrow{*} s_1 g + s_2 x$. By definition of a compatible termination order, $h' + h'$, $r_1 f + r_2 x$ and $s_1 g + s_2 x$ are smaller than u. The existence of f' and g' together with their coming arrows are consequences of Lemma 3.3. The existence of h'' and its coming arrows are due to the induction hypothesis. □

Our diamond lemma asserts that the confluence property has to be checked over subsets of syzygies instead of all the syzygies in the case where the ring $\mathbf{R}$ is a domain, that is $rs = 0$ if and only if $r = 0$ or $s = 0$. These subsets are generating set for an $\mathbf{R}$-module structure over syzygies given by the following operations:

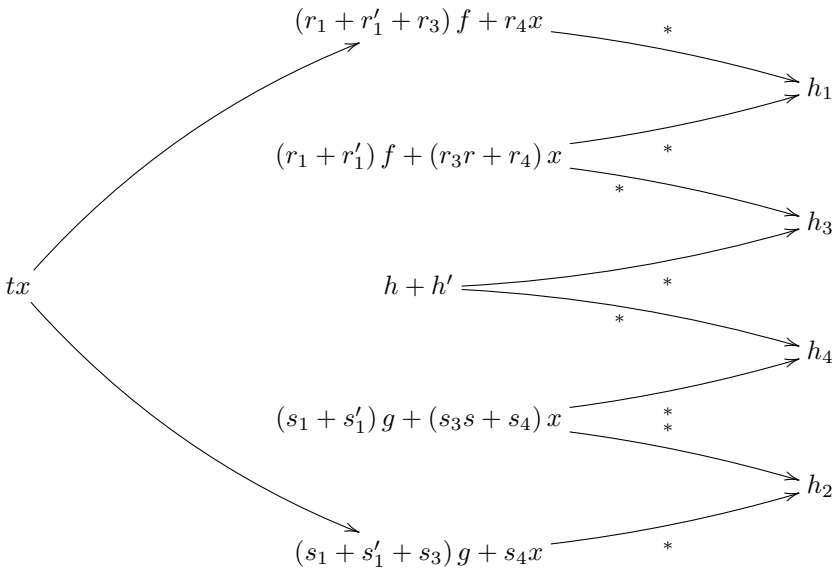
i. let $\mathbf{syz}_1 = (r_1,\ r_2,\ s_1,\ s_2)$ and $\mathbf{syz}_2 = (r'_1,\ r'_2,\ s'_1,\ s'_2)$ be two syzygies of p. We write $r_2 + r'_2 = r_3 r + r_4$ and $s_2 + s'_2 = s_3 s + s_4$. Then, we get a new syzygy $\mathbf{syz}_1 + \mathbf{syz}_2 = (r_1 + r'_1 + r_3,\ r_4,\ s_1 + s'_1 + s_3,\ s_4)$ since we have $(r_1 + r'_1 + r_3)\, r + r_4 = (r_1 + r'_1)\, r + (r_2 + r'_2) = (s_1 + s'_1)\, s + (s_2 + s'_2) = (s_1 + s'_1 + s_3)\, s + s_4$.

ii. Let $\mathbf{syz} = (r_1,\ r_2,\ s_1,\ s_2)$ be a syzygy of p and let $t \in \mathbf{R}$. We write $t r_2 = r_3 r + r_4$ and $t s_2 = s_3 s + s_4$. Then, we get a new syzygy $t\mathbf{syz} = (t r_1 + r_3,\ r_4,\ t s_1 + s_3,\ s_4)$ since we have $(t r_1 + r_3)\, r + r_4 = t\,(r_1 r + r_2) = t\,(s_1 s + s_2) = (t s_1 + s_3)\, s + s_4$.

Remark 4.3. If $\mathbf{R}$ is not a domain, then the element r_3 in **i.** and **ii.** is not unique, so that the sum of two syzygies and the product of a syzygy by an element of $\mathbf{R}$ is not well-defined.

Lemma 4.4. *Assume that* $\mathbf{R}$ *is a domain. Let* $p = (rx \longrightarrow f,\ sx \longrightarrow g)$, *let* $\mathbf{syz}_1 = (r_1,\ r_2,\ s_1,\ s_2)$ *and* $\mathbf{syz}_2 = (r'_1,\ r'_2,\ s'_1,\ s'_2)$ *be two confluent syzygies of* p *and let* $t \in \mathbf{R}$. *If* $\longrightarrow$ *is confluent at* $(r_1 + r'_1 + r_3)\, r + r_4 = (s_1 + s'_1 + s_3)\, s + s_4$ *(respectively* $(tr_1 + r_3)\, r + r_4) = (ts_1 + s_3)\, s + s_4$*), then* $\mathbf{syz}_1 + \mathbf{syz}_2$ *(respectively* $t\mathbf{syz}_1$*) is confluent.*

Sketch of proof. We only show that the sum of two confluent syzygies $\mathbf{syz}_1 + \mathbf{syz}_2$ is confluent. Let $h,\ h' \in \mathbf{R}X$ such that $r_1 f + r_2 x \xrightarrow{*} h \xleftarrow{*} s_1 g + s_2 x$ and $r'_1 f + r'_2 x \xrightarrow{*} h' \xleftarrow{*} s'_1 g + s'_2 x$. Letting $t = (r_1 + r'_1 + r_3)\, r + r_4 = (s_1 + s'_1 + s_3)\, s + s_4$, we have the following diagram:



The elements h_1, h_2 and their coming arrows are constructed using hypothesis (3). The elements h_3 and h_4 and their coming arrows are constructed using that $\longrightarrow$ is confluent at tx and Lemma 3.3. Finally, using again an inductive argument of confluence, we close the diagram and deduce that $\mathbf{syz}_1 + \mathbf{syz}_2$ is confluent.

Using similar arguments, we show that $t\mathbf{syz}_1$ is confluent, which concludes the proof. □

An adaptation of the proof of Theorem 4.2 using Lemma 4.4 provides our diamond lemma, formulated as follows:

Theorem 4.5. *Assume that* $\mathbf{R}$ *is a domain and that for every* $p = (rx \longrightarrow f,\ sx \longrightarrow g)$, *every element of a generating set of* $\mathbf{syz}(p)$ *is confluent. The rewrite relation* $\longrightarrow$ *is confluent.*

Example 4.6. Assume that $\mathbf{R}$ is a commutative field $\mathbb{K}$. For every $p = (\lambda x \longrightarrow f,\ \mu x \longrightarrow g)$, $\mathbf{syz}\,(p)$ is the vector space spanned by $(1/\lambda,\ 0,\ 1/\mu,\ 0)$. From Corollary 4.5, $\longrightarrow$ is confluent if and only if for every pair of rewrite rules $(\lambda x \longrightarrow f,\ \mu x \longrightarrow g)$, we have $f/\lambda \downarrow g/\mu$, which is equivalent to each $x \in X$ admits a unique normal form.

References

[1] David J. Anick. On the homology of associative algebras. *Trans. Amer. Math. Soc.*, 296(2):641–659, 1986.

[2] Roland Berger. Confluence and Koszulity. *J. Algebra*, 201(1):243–283, 1998.

[3] Roland Berger. Koszulity for nonquadratic algebras. *J. Algebra*, 239(2):705–734, 2001.

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

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

[6] Yuji Kobayashi. Gröbner bases of associative algebras and the Hochschild cohomology. *Trans. Amer. Math. Soc.*, 357(3):1095–1124 (electronic), 2005.

[7] 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).

[8] Lionel Vaux. The algebraic lambda calculus. *Math. Structures Comput. Sci.*, 19(5):1029–1059, 2009.

Certified Ordered Completion*

Christian Sternagel and Sarah Winkler

Department of Computer Science
University of Innsbruck, Innsbruck, Austria
{christian.sternagel|sarah.winkler}@uibk.ac.at

Abstract

On the one hand, ordered completion is a fundamental technique in equational theorem proving that is employed by automated tools. On the other hand, their complexity makes such tools inherently error prone. As a remedy to this situation we give an Isabelle/HOL formalization of ordered rewriting and completion that comes with a formally verified certifier for ordered completion proofs. By validating generated proof certificates, our certifier increases the reliability of ordered completion tools.

1 Introduction

Completion has evolved as a fundamental technique in automated reasoning since the groundbreaking work by Knuth and Bendix [5]. Its goal is to transform a given set of equations into a terminating and confluent term rewrite system that induces the same equational theory and can thus be used to decide equivalence with respect to the initial set of equations. Since the original procedure can fail if unorientable equations are encountered, ordered completion was developed to remedy this shortcoming [2]. The systems generated by ordered completion tools are in general only ground confluent, but this turns out to be sufficient for practical applications like refutational theorem proving.

Consider for example the following equational system $\mathcal{E}_0$ which the tool MædMax [10]

$$\begin{array}{ccc} x \div y \approx \langle 0, 0\rangle & x \div y \approx \langle \mathsf{s}(q), \mathsf{s}(q)\rangle & x - 0 \approx x \\ 0 - y \approx 0 & \mathsf{s}(x) - \mathsf{s}(y) \approx x - y & \mathsf{s}(x) > \mathsf{s}(y) \approx x > y \\ \mathsf{s}(x) > 0 \approx \mathsf{true} & \mathsf{s}(x) \leq \mathsf{s}(y) \approx x \leq y & 0 \leq x \approx \mathsf{true} \end{array}$$

transforms by ordered completion into the following rules $\mathcal{R}$ ($\to$) and equations $\mathcal{E}$ ($\approx$):

$$\begin{array}{cccc} x - 0 \to x & 0 - x \to 0 & \mathsf{s}(x) - \mathsf{s}(y) \to x - y & x \div y \to \langle 0, 0\rangle \\ 0 \leq x \to \mathsf{true} & \mathsf{s}(x) \leq \mathsf{s}(y) \to x \leq y & \mathsf{s}(x) > 0 \to \mathsf{true} & \\ \mathsf{s}(x) > \mathsf{s}(y) \to x > y & \langle \mathsf{s}(x), \mathsf{s}(x)\rangle \approx \langle \mathsf{s}(q), \mathsf{s}(q)\rangle & \langle \mathsf{s}(q), \mathsf{s}(q)\rangle \approx \langle 0, 0\rangle & \langle 0, 0\rangle \approx \langle 0, 0\rangle \end{array}$$

This system can be used to decide a given ground equation by checking whether the terms' unique normal forms (with respect to ordered rewriting) are equal.

Such ground complete systems are useful for other tools, like ConCon [9]—a tool for automatically proving confluence of conditional term rewrite systems—which employs ordered completion for proving infeasibility of conditional critical pairs. In fact, $\mathcal{E}_0$ from our initial example is the equational system that ConCon derives from Cops #361 for that purpose. The latter models division with remainder, though the transformation performed by ConCon creates some equations which do not fit into this semantics but are required to decide confluence.

*This work is supported by the Austrian Science Fund (FWF): projects T789 and P27502.

However, automated tools like ConCon and MædMax are complex and highly optimized. The produced proofs often comprise hundreds of equations and thousands of steps. Hence care should be taken to trust the output of such tools.

To improve this situation we follow a two-staged certification approach and first (1) add the relevant concepts and results to a formal library, and then (2) use code generation to obtain a trusted certifier. More specifically, our contributions are as follows:

- Regarding stage (1), we extended the ***Isa**belle **F**ormalization **o**f **R**ewriting*[1] (IsaFoR) by ordered rewriting and a generalization of the ordered completion calculus oKB [2], and proved the latter correct for finite runs using ground-total reduction orders (Section 3). Moreover, we established ground-totality of the lexicographic path order and the Knuth-Bendix order.
- With respect to stage (2), we extended the XML-based *certification problem format* (CPF for short) [8] by certificates comprising the initial equations, the resulting system along with a reduction order, and a stepwise derivation of the latter from the former. We then formalized check functions that verify that the supplied derivation corresponds to a valid oKB run whose final state matches the resulting system (Section 4). As a result CeTA (the certifier accompanying IsaFoR) can now certify ordered completion proofs produced by the tool MædMax [10].

2 Preliminaries

In the sequel we use standard notation from term rewriting [1]. We consider the *set of all terms* $\mathcal{T}(\mathcal{F},\mathcal{V})$ over a signature $\mathcal{F}$ and an infinite set of variables $\mathcal{V}$, while $\mathcal{T}(\mathcal{F})$ denotes the *set of all ground terms*. A *substitution* σ is a mapping from variables to terms. As usual, we write $t\sigma$ for the *application* of σ to a term t. A *variable permutation* (or *renaming*) π is a bijective substitution such that $\pi(x) \in \mathcal{V}$ for all $x \in \mathcal{V}$. For an equational system (ES) $\mathcal{E}$ we write $\mathcal{E}^{\leftrightarrow}$ to denote its symmetric closure $\mathcal{E} \cup \{t \approx s \mid s \approx t \in \mathcal{E}\}$. For a reduction order $>$ and an ES $\mathcal{E}$, the term rewrite system (TRS) $\mathcal{E}^{>}$ consists of all rules $s\sigma \to t\sigma$ such that $s \approx t \in \mathcal{E}$ and $s\sigma > t\sigma$.

Given a reduction order $>$, an *extended overlap* is given by two variable-disjoint variants $\ell_1 \approx r_1$ and $\ell_2 \approx r_2$ of equations in $\mathcal{E}^{\leftrightarrow}$ such that $p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(\ell_2)$ and ℓ_1 and $\ell_2|_p$ are unifiable with most general unifier μ. An extended overlap which in addition satisfies $r_1\mu \not> \ell_1\mu$ and $r_2\mu \not> \ell_2\mu$ gives rise to the *extended critical pair* $\ell_2[r_1]_p\mu \approx r_2\mu$. The set $\mathrm{CP}_{>}(\mathcal{E})$ consists of all extended critical pairs among equations in $\mathcal{E}$. A TRS $\mathcal{R}$ is *(ground) complete* if it is terminating and confluent (on ground terms). Finally, we say that a TRS $\mathcal{R}$ is a presentation of an ES $\mathcal{E}$, whenever $\leftrightarrow^*_{\mathcal{E}} = \leftrightarrow^*_{\mathcal{R}}$.

3 Formalizing Ordered Completion

We consider the following definition of ordered completion.

Definition 1 (Ordered Completion)**.** The inference system oKB of ordered completion operates on pairs $(\mathcal{E},\mathcal{R})$ of equations $\mathcal{E}$ and rules $\mathcal{R}$ over a common signature $\mathcal{F}$. It consists of the following inference rules, where $\mathcal{S}$ abbreviates $\mathcal{R} \cup \mathcal{E}^{>}$ and π is a renaming.

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

$$\textsf{deduce}\ \frac{\mathcal{E},\mathcal{R}}{\mathcal{E}\cup\{s\pi\approx t\pi\},\mathcal{R}}\ \text{if } s \xleftarrow[\mathcal{R}\cup\mathcal{E}]{} \cdot \xrightarrow[\mathcal{R}\cup\mathcal{E}]{} t \qquad \textsf{compose}\ \frac{\mathcal{E},\mathcal{R}\uplus\{s\to t\}}{\mathcal{E},\mathcal{R}\cup\{s\pi\to u\pi\}}\ \text{if } t\to_{\mathcal{S}} u$$

$$\textsf{orient}\ \frac{\mathcal{E}\uplus\{s\approx t\},\mathcal{R}}{\mathcal{E},\mathcal{R}\cup\{s\pi\to t\pi\}}\ \text{if } s>t \qquad \textsf{simplify}\ \frac{\mathcal{E}\uplus\{s\approx t\},\mathcal{R}}{\mathcal{E}\cup\{u\pi\approx t\pi\},\mathcal{R}}\ \text{if } s\to_{\mathcal{S}} u$$

$$\frac{\mathcal{E}\uplus\{s\approx t\},\mathcal{R}}{\mathcal{E},\mathcal{R}\cup\{t\pi\to s\pi\}}\ \text{if } t>s \qquad \frac{\mathcal{E}\uplus\{s\approx t\},\mathcal{R}}{\mathcal{E}\cup\{s\pi\approx u\pi\},\mathcal{R}}\ \text{if } t\to_{\mathcal{S}} u$$

$$\textsf{delete}\ \frac{\mathcal{E}\uplus\{s\approx s\},\mathcal{R}}{\mathcal{E},\mathcal{R}} \qquad \textsf{collapse}\ \frac{\mathcal{E},\mathcal{R}\uplus\{t\to s\}}{\mathcal{E}\cup\{u\pi\approx s\pi\},\mathcal{R}}\ \text{if } t\to_{\mathcal{S}} u$$

We write $(\mathcal{E},\mathcal{R})\vdash(\mathcal{E}',\mathcal{R}')$ if $(\mathcal{E}',\mathcal{R}')$ is obtained from $(\mathcal{E},\mathcal{R})$ by employing one of the above inference rules. A finite sequence of inferences $(\mathcal{E}_0,\varnothing)\vdash(\mathcal{E}_1,\mathcal{R}_1)\vdash\cdots\vdash(\mathcal{E}_n,\mathcal{R}_n)$ is called a *run*. Definition 1 differs from the original formulation of ordered completion [2] in two ways. First, collapse and simplify do not require an encompassment condition. This omission is possible since we only consider *finite* runs. Second, we allow variants of rules and equations to be added. This relaxation tremendously simplifies certificate generation in tools, where facts are renamed upon generation to avoid the maintenance and processing of many renamed versions of one equation.

The following inclusions express straightforward properties of oKB.

Lemma 1. *If $(\mathcal{E},\mathcal{R})\vdash^*(\mathcal{E}',\mathcal{R}')$ then $\mathcal{R}\subseteq{>}$ implies $\mathcal{R}'\subseteq{>}$.* □

Lemma 2. *If $(\mathcal{E},\mathcal{R})\vdash^*(\mathcal{E}',\mathcal{R}')$ then the conversion equivalence $\leftrightarrow^*_{\mathcal{E}\cup\mathcal{R}}=\leftrightarrow^*_{\mathcal{E}'\cup\mathcal{R}'}$ holds.* □

The following abstract result is the key ingredient to our proof of ground completeness.

Lemma 3. *Let $\mathcal{E}$ be an ES and $>$ a reduction order such that $s>t$ or $t\approx s\in\mathcal{E}$ holds for all $s\approx t\in\mathcal{E}$. If for all $s\approx t\in \mathrm{CP}_{>}(\mathcal{E})$ we have $s\downarrow_{\mathcal{E}^>}t$ or there is some $s'\approx t'\in\mathcal{E}^{\leftrightarrow}$ such that $s\approx t=(s'\approx t')\sigma$ then $\mathcal{E}^>$ is ground complete.* □

In combination, Lemmas 1, 2, and 3 allow us to obtain our main correctness result: acceptance of a certificate by our check function implies that $\mathcal{R}\cup\mathcal{E}^>$ is a ground complete presentation of $\mathcal{E}_0$. For simplicity's sake, we give only the corresponding high-level result (that is, not mentioning our concrete implementation):

Theorem 1. *Suppose $(\mathcal{E}_0,\varnothing)\vdash^*(\mathcal{E},\mathcal{R})$ was obtained using a ground-total reduction order $>$ with minimal constant c and for all $s\approx t\in\mathrm{CP}_{>}(\mathcal{E}^{\leftrightarrow}\cup\mathcal{R})$ either $s\downarrow_{\mathcal{R}\cup\mathcal{E}^>}t$, or $s\approx t=(s'\approx t')\sigma$ for some $s'\approx t'\in\mathcal{E}^{\leftrightarrow}$. Then $\leftrightarrow^*_{\mathcal{E}_0}=\leftrightarrow^*_{\mathcal{R}\cup\mathcal{E}}$ and $\mathcal{R}\cup\mathcal{E}^>$ is ground complete.* □

This result employs the following sufficient condition for ground completeness: all critical pairs are joinable or instances of equations already present. In fact, this is not a necessary condition. Martin and Nipkow [6] gave examples of ground confluent systems that do not satisfy this condition, and presented a stronger criterion. However, ground confluence is known to be undecidable even for terminating TRSs [4], hence no complete criterion can be implemented.

Ground-total reduction orders. Ground confluence crucially relies on ground-total reduction orders. Our IsaFoR proofs of the following results follow the standard textbook approach [1].

Lemma 4. *If $>$ is a total precedence on $\mathcal{F}$ then $>_{\mathsf{lpo}}$ is total on $\mathcal{T}(\mathcal{F})$.* □

Lemma 5. *If $>$ is a total precedence on $\mathcal{F}$ then $>_{\mathsf{kbo}}$ is total on $\mathcal{T}(\mathcal{F})$.* □

In addition, we proved that for any given KBO $>_{\mathsf{kbo}}$ (LPO $>_{\mathsf{lpo}}$) defined over a total precedence $>$ there exists a minimal constant c such that $t\geqslant_{\mathsf{kbo}}c$ ($t\geqslant_{\mathsf{lpo}}c$) holds for all $t\in\mathcal{T}(\mathcal{F})$.

4 Checking Ordered Completion Proofs

While CeTA has supported certification of standard completion for quite some time [7], certification of ordered completion proofs is considerably more intricate. For standard completion, the certificate contains the initial set of equations $\mathcal{E}_0$, the resulting TRS $\mathcal{R}$ together with a termination proof, and stepwise $\mathcal{E}_0$-conversions from ℓ to r for each rule $\ell \to r \in \mathcal{R}$. The certifier first checks the termination proof to guarantee termination of $\mathcal{R}$. This allows us to establish confluence of $\mathcal{R}$ by ensuring that all critical peaks are joinable. At this point it is easy to verify $\leftrightarrow^*_{\mathcal{E}_0} \subseteq \leftrightarrow^*_{\mathcal{R}}$: for each equation $s \approx t \in \mathcal{E}_0$ compute the $\mathcal{R}$-normal forms of s and t and check for syntactic equality. The converse inclusion $\leftrightarrow^*_{\mathcal{R}} \subseteq \leftrightarrow^*_{\mathcal{E}_0}$ is taken care of by the provided $\mathcal{E}_0$-conversions. Overall, we obtain that $\mathcal{R}$ is a complete presentation of $\mathcal{E}_0$ without mentioning a specific inference system for completion.

Unfortunately, the same approach does not work for ordered completion: The inclusion $\leftrightarrow^*_{\mathcal{E}_0} \subseteq \leftrightarrow^*_{\mathcal{R}\cup\mathcal{E}}$ cannot be established by rewriting equations in $\mathcal{E}_0$ to normal form, since they may contain variables but $\mathcal{R} \cup \mathcal{E}^>$ is only ground confluent. Therefore, we instead ask for certificates that contain the input equalities $\mathcal{E}_0$, the resulting equations and rules $(\mathcal{E}, \mathcal{R})$, the reduction order $>$, and a sequence of inference steps according to Definition 1. A valid certificate ensures (by Lemma 2) that the relations $\leftrightarrow^*_{\mathcal{E}_0}$ and $\leftrightarrow^*_{\mathcal{R}\cup\mathcal{E}}$ coincide.

The certificate corresponding to our initial example contains the equations $\mathcal{E}_0$, the resulting system $(\mathcal{E}, \mathcal{R})$, and the reduction order $>_{\mathsf{kbo}}$ with precedence $> > \mathsf{s} > \leq > \mathsf{true} > - > \div > \mathsf{p} > 0$, $w_0 = 1$, and $w(0) = 2$, $w(\div) = w(\mathsf{true}) = w(\mathsf{s}) = 1$, and all other symbols having weight 0. In addition, a sequence of inference steps explains how $(\mathcal{E}, \mathcal{R})$ is obtained from $\mathcal{E}_0$:

$\mathsf{simplify}_{\mathsf{left}}$	$x \div y \approx \langle \mathsf{s}(q), \mathsf{s}(q)\rangle$ to $\langle 0, 0\rangle \approx \langle \mathsf{s}(q), \mathsf{s}(q)\rangle$
deduce	$\langle 0, 0\rangle \leftarrow \langle \mathsf{s}(u), \mathsf{s}(u)\rangle \to \langle 0, 0\rangle$
deduce	$\langle \mathsf{s}(x), \mathsf{s}(x)\rangle \leftarrow \langle 0, 0\rangle \to \langle \mathsf{s}(q), \mathsf{s}(q)\rangle$
deduce	$x > y \leftarrow \mathsf{s}(x) > \mathsf{s}(y) \to \mathsf{s}(\mathsf{s}(x)) > \mathsf{s}(\mathsf{s}(y))$
deduce	$\mathsf{s}(\mathsf{s}(x)) > \mathsf{s}(0) \leftarrow \mathsf{s}(x) > 0 \to \mathsf{true}$
$\mathsf{orient}_{\mathsf{rl}}$	$0 \leq x \to \mathsf{true}$
$\mathsf{orient}_{\mathsf{lr}}$	$\mathsf{s}(\mathsf{s}(x)) > \mathsf{s}(0) \to \mathsf{true}$
$\mathsf{orient}_{\mathsf{rl}}$	$\mathsf{s}(x) > \mathsf{s}(y) \to x > y$
$\mathsf{orient}_{\mathsf{lr}}$	$\mathsf{s}(x) > 0 \to \mathsf{true}$
$\mathsf{orient}_{\mathsf{rl}}$	$\mathsf{s}(\mathsf{s}(x)) > \mathsf{s}(\mathsf{s}(y)) \to x > y$
$\mathsf{orient}_{\mathsf{rl}}$	$x - 0 \to x$
$\mathsf{orient}_{\mathsf{lr}}$	$x \div y \to \langle 0, 0\rangle$
$\mathsf{orient}_{\mathsf{rl}}$	$\mathsf{s}(x) - \mathsf{s}(y) \to x - y$
$\mathsf{orient}_{\mathsf{rl}}$	$0 - x \to 0$
$\mathsf{orient}_{\mathsf{rl}}$	$\mathsf{s}(x) \leq \mathsf{s}(y) \to x \leq y$
collapse	$\mathsf{s}(\mathsf{s}(x)) > \mathsf{s}(\mathsf{s}(y)) \to x > y$ to $\mathsf{s}(x) > \mathsf{s}(y) \approx x > y$
$\mathsf{simplify}_{\mathsf{left}}$	$\mathsf{s}(x) > \mathsf{s}(y) \approx x > y$ to $x > y \approx x > y$
collapse	$\mathsf{s}(\mathsf{s}(x)) > \mathsf{s}(0) \to \mathsf{true}$ to $\mathsf{s}(x) > 0 \approx \mathsf{true}$
$\mathsf{simplify}_{\mathsf{left}}$	$\mathsf{s}(x) > 0 \approx \mathsf{true}$ to $\mathsf{true} \approx \mathsf{true}$
delete	$x > y \approx x > y$
delete	$\mathsf{true} \approx \mathsf{true}$

Given such a certificate, CeTA checks that the provided sequence of inferences forms a run $(\mathcal{E}_0\pi, \varnothing) \vdash^* (\mathcal{E}, \mathcal{R})$ for some renaming π. Verifying the validity of individual inferences involves checking side conditions such as orientability of a term pair in an orient step with respect to the given reduction order. Then it is checked that $\mathcal{R} \cup \mathcal{E}^>$ is ground confluent according to the criterion of Theorem 1. Finally, it is ensured that the given reduction order $>$ has a total

precedence (and is admissible, in the case of KBO). As usual in CeTA, error messages are printed if one of these checks fails, pointing out the reason for the proof being rejected.

5 Conclusion

We presented our formalization of ordered completion in IsaFoR, which enables CeTA (starting with version 2.33) to certify ordered completion proofs. To the best of our knowledge, CeTA thus constitutes the first formally verified certifier for ordered completion.

Together with Hirokawa and Middeldorp we reported on another Isabelle/HOL formalization of ordered completion [3]. The main difference to our current work is that this other formalization is based on a more restrictive inference system of ordered completion that also covers infinite runs, while we restrict to finite runs in the interest of certification. Indeed every finite run akin to [3, Definition 18] is also a run according to Definition 1, while the inference sequence in our running example is not possible in the former setting.

As future work, we plan to add more powerful criteria for ground confluence to IsaFoR, and support equational disproofs based on ground complete systems in CeTA. To that end, it would be useful to also support narrowing in CeTA. Certified equational disproofs could in turn be used to certify confluence proofs by ConCon which rely on infeasibility of conditional critical pairs.

References

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

[2] L. Bachmair, N. Dershowitz, and D. A. Plaisted. Completion without failure. In H. A. Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 2 of *Rewriting Techniques*, pages 1–30. Academic Press, 1989. doi:10.1016/B978-0-12-046371-8.50007-9.

[3] N. Hirokawa, A. Middeldorp, C. Sternagel, and S. Winkler. Infinite runs in abstract completion. In *Proc. 2nd FSCD*, volume 84 of *LIPIcs*, pages 19:1–19:16, 2017. doi:10.4230/LIPIcs.FSCD.2017.19.

[4] D. Kapur, P. Narendran, and F. Otto. On ground-confluence of term rewriting systems. *Inf. Comput*, 86(1):14–31, 1990. doi:10.1016/0890-5401(90)90023-B.

[5] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970. doi:10.1016/B978-0-08-012975-4.

[6] U. Martin and T. Nipkow. Ordered Rewriting and Confluence. In *Proc. 10th CADE*, volume 449 of *LNCS*, pages 366–380, 1990. doi:10.1007/3-540-52885-7_100.

[7] C. Sternagel and R. Thiemann. Formalizing Knuth-Bendix orders and Knuth-Bendix completion. In *Proc. 24th RTA*, volume 21 of *LIPIcs*, pages 287–302, 2013. doi:10.4230/LIPIcs.RTA.2013.287.

[8] C. Sternagel and R. Thiemann. The certification problem format. In *Proc. 11th UITP*, volume 167 of *EPTCS*, pages 61–72, 2014. doi:10.4204/EPTCS.167.8.

[9] T. Sternagel and A. Middeldorp. Conditional confluence (system description). In *Proc. RTA/TLCA 2014*, volume 8560 of *LNCS*, pages 456–465, 2014. doi:10.1007/978-3-319-08918-8_31.

[10] S. Winkler and G. Moser. Maedmax: A maximal ordered completion tool. In *Proc. 9th IJCAR*, 2018. To appear.

Towards a Verified Decision Procedure for Confluence of Ground Term Rewrite Systems in Isabelle/HOL*

Bertram Felgenhauer[1] Aart Middeldorp[1]
T. V. H. Prathamesh[1] Franziska Rapp[1]

Department of Computer Science, University of Innsbruck, Austria
{bertam.felgenhauer,aart.middeldorp,venkata.turaga,franziska.rapp}@uibk.ac.at

Abstract

Confluence is a decidable property of ground rewrite systems. We present a formalization effort in Isabelle/HOL of the decision procedure based on ground tree transducers.

1 Introduction

Confluence is an undecidable property of term rewrite systems. Oyamaguchi [7] was the first to prove the decidability of confluence for *ground* rewrite systems. Dauchet, Heuillard, Lescanne, and Tison [3] presented a simpler decidability proof based on *ground tree transducers*. Comon, Godoy, and Nieuwenhuis [2] were the first to prove that confluence of ground rewrite systems is decidable in polynomial time and Felgenhauer [6] presented a cubic time algorithm.

In [4] the decision procedure of [3] was extended to *left-linear, right-ground* rewrite systems. Dauchet and Tison [5] showed that the *first-order theory* of rewriting is decidable for ground rewrite systems. In this theory properties definable by a first-order formula over rewrite predicates like $\to$ and $\to^*$ are expressible. This includes confluence. The decision procedure (extended to left-linear, right-ground rewrite systems) is implemented in FORT [8]. Ground tree transducers and their closure properties play a key role in the decision procedure.

Our long-term aim is to formalize the decision procedure in the proof assistant Isabelle/HOL such that the output of FORT can be certified. (To this end, FORT would emit a sequence of operations on automata that correspond to a formula; the certifier would then compute the corresponding automata using a verified implementation.) In this paper we present a formalization of ground tree transducers and their closure properties. Furthermore, a number of results on the interplay between rewriting and ground tree transducers are formalized, bringing us close to the first formalized proof of the decidability of confluence of ground rewrite systems.

Our formalization is based on IsaFoR [9][1]. Our own development can be found at http://cl-informatik.uibk.ac.at/software/fortissimo/iwc2018/. Furthermore most definitions, theorems, and lemmas directly correspond to the formalization. These are indicated by the ✔ symbol, which links to a HTML presentation in the PDF version of the paper.

2 Preliminaries

We assume familiarity with term rewriting and (bottom-up) tree automata. Let $\mathcal{R}$ be a ground term rewrite system (TRS for short) over a signature $\mathcal{F}$, where $\mathcal{F}$ contains at least one constant (which is assured if $\mathcal{R} \neq \varnothing$.) A tree automaton $\mathcal{A} = (Q, Q_f, \Delta)$ consists of a set of states

*This work is supported by the Austrian Science Fund (FWF): project P30301.
[1] http://cl-informatik.uibk.ac.at/isafor

Q, a set of final states Q_f, and a set of transitions Δ. Ordinary transitions have the form $f(q_1, \ldots, q_n) \to q$ where $q_1, \ldots, q_n$ and q are states, and $f \in \mathcal{F}$ has arity n, while ϵ-transitions $p \to q$ are between states. Noting that the transitions are ground rewrite rules, we write $\to_{\mathcal{A}}$ for $\to_{\Delta}$. To decide confluence of $\mathcal{R}$, first a ground tree transducer (GTT for short) $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ is constructed that recognizes a relation in between[2] $\to_{\mathcal{R}}$ and $\to^*_{\mathcal{R}}$. A GTT $\mathcal{G}$ consists of two tree automata $\mathcal{A}$ and $\mathcal{B}$ that operate on the same signature. A pair of ground terms s and t is accepted by $\mathcal{G}$ if $s \to^*_{\mathcal{A}} \cdot {}^*_{\mathcal{B}}\!\leftarrow t$; we denote the relation consisting of all such pairs (s, t) by $\mathcal{L}(\mathcal{G})$.

Next the transitive closure $\mathcal{G}^*$ of $\mathcal{G}$ is computed by an iterative procedure in which certain ϵ-transitions are added to the involved tree automata. Since $(\to^*_{\mathcal{R}})^* = \to^*_{\mathcal{R}}$, the GTT $\mathcal{G}^*$ recognizes reachability. The relation ${}^*_{\mathcal{R}}\!\leftarrow$ is recognized by the inverse $\mathcal{G}^{*-}$ of $\mathcal{G}^*$ (which is simply obtained by interchanging the two tree automata that make up $\mathcal{G}^*$).

The GTTs $\mathcal{G}^*$ and $\mathcal{G}^{*-}$ are composed to obtain GTTs $\mathcal{G}_1$ and $\mathcal{G}_2$ that recognize the relations $\uparrow_{\mathcal{R}} = {}^*_{\mathcal{R}}\!\leftarrow \cdot \to^*_{\mathcal{R}}$ and $\downarrow_{\mathcal{R}} = \to^*_{\mathcal{R}} \cdot {}^*_{\mathcal{R}}\!\leftarrow$. The final step of the decision procedure is the inclusion check $\mathcal{L}(\mathcal{G}_1) \subseteq \mathcal{L}(\mathcal{G}_2)$. In [4] this is done by applying an ad-hoc recognizability preserving transformation from GTTs to tree automata over an extended signature, and subsequently using a decision procedure for tree language inclusion. In our formalization we instead associate RR_2 automata to $\mathcal{G}_1$ and $\mathcal{G}_2$, followed by an inclusion check for RR_n automata, where RR_n relations are a way of capturing n-ary relations on terms as regular tree languages [1]. The reason for this is that RR_n automata play a key role in the decision procedure for the first-order theory of rewriting. Therefore we can reuse our results when formalizing further aspects of the theory implemented in FORT. On the other hand, the detour via GTTs is necessary because RR_2 relations are not closed under transitive closure.

The most complicated part of the above procedure is the closure of GTT relations under composition and transitive closure. Proofs of these results are presented in detail in [1, Section 3.2]. Below we present (simpler) paper proofs that correspond to our formalization.

3 Formalizing the Confluence Check

We rely on IsaFoR's formalization of tree automata, where a tree automaton is a triple consisting of the set of final states (which is irrelevant for GTTs), the set of ordinary transitions, and the set of epsilon transitions. The set of states of the automaton is left implicit. For example,

$$ta.make\ \{0\}\ \{\mathsf{a}\ [] \to 1, \mathsf{f}\ [1] \to 0\}\ \{(0, 1)\}$$

would be an automaton that accepts $\mathsf{f}^k(\mathsf{a})$ for $k \geqslant 1$ (the transitions are $\mathsf{a} \to 1$, $\mathsf{f}(1) \to 0$, and $0 \to 1$). Note that ordinary transitions and ϵ-transitions have different types, hence different notation. We can check whether an automaton $\mathcal{A}$ accepts a term t in state q using $q \in ta\_res\ \mathcal{A}\ t$. The language accepted by $\mathcal{A}$ is provided as $ta\_lang\ \mathcal{A}$. GTTs are formalized as pairs of tree automata with the same state and function symbol types. The relation accepted by a GTT is formalized by the predicate *gtt_accept*, which is equivalent ✔ to *gtt_accept′* given in Listing 1.

The first step of the construction is to obtain a GTT from the given ground TRS $\mathcal{R}$. To this end, we follow the construction of Dauchet et al. [4]. Let $\langle s \rangle$ be a state for each subterm $s \trianglelefteq \mathcal{R}$ (meaning there is a rule $l \to r$ in $\mathcal{R}$ such that $s \trianglelefteq l$ or $s \trianglelefteq r$). Let $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ where

$$\Delta_{\mathcal{A}} = \{f(\langle t_1 \rangle, \ldots, \langle t_n \rangle) \to \langle f(t_1, \ldots, t_n) \rangle \mid f(t_1, \ldots, t_n) \trianglelefteq \mathcal{R}\} \cup \{\langle l \rangle \to \langle r \rangle \mid l \to r \in \mathcal{R}\}$$

and $\Delta_{\mathcal{B}}$ is defined symmetrically (replacing $\langle l \rangle \to \langle r \rangle$ by $\langle r \rangle \to \langle l \rangle$ in the second subset). In the

[2] It is also true that $\twoheadrightarrow_{\mathcal{R}}$ is recognizable by a GTT ✔ ✔ but for confluence this weaker result suffices.

```
inductive gtt_accept' :: ('q, 'f) gtt ⇒ ('f, 'q) term ⇒ ('f, 'q) term ⇒ bool
  for 𝒢 where
  mctxt [intro]: length ss = length ts ⟹ num_holes C = length ss ⟹
    ∀ i < length ts. ∃ q. q ∈ ta_res (fst 𝒢) (ss ! i) ∧ q ∈ ta_res (snd 𝒢) (ts ! i) ⟹
    gtt_accept' 𝒢 (fill_holes C ss) (fill_holes C ts)
```

Listing 1: Definition of GTT acceptance.

```
definition cmn_ta_rules::('f, 'v) term set ⇒ (('f, 'v) term option, 'f) ta_rule set
where
  cmn_ta_rules T =
    {(f (map Some ts)) → Some (Fun f ts)) |f ts t. Fun f ts ◁ t ∧ t ∈ T}

definition trs_to_ta_A::('f, 'v) trs ⇒ (('f, 'v) term option, 'f) ta where
  trs_to_ta_A R = ta.make {} (cmn_ta_rules (TRS_terms R))
    {(Some l, Some r) |l r. (l,r) ∈ R}
```

Listing 2: Associating a GTT to a TRS.

formalization, $\langle s \rangle$ is represented by *Some* s.[3] This gives rise to the definitions in Listing 2. The resulting GTT is suitable for simulating sequences of $\mathcal{R}$ steps, by the following theorem.

Theorem 1. $\to_{\mathcal{R}} \subseteq \mathcal{L}(\mathcal{A}, \mathcal{B})$ ✔ *and* $\mathcal{L}(\mathcal{A}, \mathcal{B}) \subseteq \to_{\mathcal{R}}^*$. ✔

Example 2. We illustrate the construction on the ground TRS $\mathcal{R}$ consisting of the rules $\mathsf{a} \to \mathsf{f}(\mathsf{a})$, $\mathsf{a} \to \mathsf{b}$, and $\mathsf{f}(\mathsf{b}) \to \mathsf{c}$. We construct the GTT $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ with Δ consisting of the rules

$$\mathsf{a} \to \langle \mathsf{a} \rangle \qquad \mathsf{b} \to \langle \mathsf{b} \rangle \qquad \mathsf{c} \to \langle \mathsf{c} \rangle \qquad \mathsf{f}(\langle \mathsf{a} \rangle) \to \langle \mathsf{f}(\mathsf{a}) \rangle \qquad \mathsf{f}(\langle \mathsf{b} \rangle) \to \langle \mathsf{f}(\mathsf{b}) \rangle$$

to recognize all subterms in the rules of $\mathcal{R}$, $\Delta_A = \Delta \cup \{\langle \mathsf{a} \rangle \to \langle \mathsf{f}(\mathsf{a}) \rangle, \langle \mathsf{a} \rangle \to \langle \mathsf{b} \rangle, \langle \mathsf{f}(\mathsf{b}) \rangle \to \langle \mathsf{c} \rangle\}$, and $\Delta_B = \Delta \cup \{\langle \mathsf{f}(\mathsf{a}) \rangle \to \langle \mathsf{a} \rangle, \langle \mathsf{b} \rangle \to \langle \mathsf{a} \rangle, \langle \mathsf{c} \rangle \to \langle \mathsf{f}(\mathsf{b}) \rangle\}$. Note that $L(\mathcal{G})$ accepts more than $\twoheadrightarrow_{\mathcal{R}}$. For instance, $(\mathsf{a}, \mathsf{f}(\mathsf{b})) \in L(\mathcal{G})$ as $\mathsf{a} \to_{\mathcal{A}}^* \langle \mathsf{f}(\mathsf{a}) \rangle \,{}_{\mathcal{B}}^*\!\leftarrow \mathsf{f}(\mathsf{b})$ but $\mathsf{a} \not\twoheadrightarrow_{\mathcal{R}} \mathsf{f}(\mathsf{b})$ does not hold.

To illustrate one of the minor (but tedious) issues that come up in the formalization, note that the state type of the GTT seeps into terms accepted by the GTT: they are objects of type ('*f*, ('*f*, '*q*) *term option*) *term*. On the other hand, $\mathcal{R}^*$ is a relation between terms of type ('*f*, '*v*) *term*, with a completely different variable type. But actually, since we deal with ground terms, the variable type does not matter. In order to express this property, we use the existent *adapt_vars* function that changes the variable type arbitrarily.

The next step in the decision procedure is the computation of the transitive closure. However, that computation builds on the composition of GTT relations, so we present that first. The composition combines the transitions of the constituent GTTs, and adds carefully chosen epsilon transitions.

Definition 3. ✔ Let $\mathcal{G}_1 = (\mathcal{A}_1, \mathcal{B}_1)$ and $\mathcal{G}_2 = (\mathcal{A}_2, \mathcal{B}_2)$ be GTTs. We let

$$GTT\_comp(\mathcal{G}_1, \mathcal{G}_2) = (\Delta_{\mathcal{A}_1} \cup \Delta_{\mathcal{A}_2} \cup \Delta_\epsilon(\mathcal{B}_1, \mathcal{A}_2), \Delta_{\mathcal{A}_1} \cup \Delta_{\mathcal{A}_2} \cup \Delta_\epsilon(\mathcal{A}_2, \mathcal{B}_1))$$

Here $\Delta_\epsilon(\mathcal{A}, \mathcal{B}) = \{(p, q) \mid t \to_{\mathcal{A}}^* p \text{ and } t \to_{\mathcal{B}}^* q \text{ for some } t \in \mathcal{T}(\mathcal{F})\}$.

[3] This use of the option type is not really necessary, but it was helpful to distinguish states and terms while developing the proofs.

This construction is simplified compared to [1,3]. Compared to [3], only ϵ-transitions are added, while [1] actually adds fewer ϵ-transitions than our definition, but at the cost of a less symmetric definition.

Example 4. Continuing Example 2, we let Q be the set of states in Δ and compute $\Delta_\epsilon(\mathcal{A},\mathcal{B}) = \mathsf{Id}_Q \cup \{\langle \mathsf{b}\rangle \to \langle \mathsf{a}\rangle, \langle \mathsf{c}\rangle \to \langle \mathsf{f}(\mathsf{b})\rangle\} \cup \{\langle \mathsf{c}\rangle, \langle \mathsf{f}(\mathsf{a})\rangle, \langle \mathsf{f}(\mathsf{b})\rangle\} \times \{\langle \mathsf{a}\rangle, \langle \mathsf{f}(\mathsf{a})\rangle\}$ and $\Delta_\epsilon(\mathcal{B},\mathcal{A}) = \Delta_\epsilon(\mathcal{A},\mathcal{B})^-$. For instance, the transition rule $\langle \mathsf{c}\rangle \to \langle \mathsf{a}\rangle \in \Delta_\epsilon(\mathcal{A},\mathcal{B})$ is witnessed by the term $\mathsf{f}(\mathsf{b})$.

Theorem 5. ✔ *If R_1 and R_2 are recognizable relations then $R_1 \circ R_2$ is a recognizable relation. More precisely, if R_1 and R_2 are recognized by $\mathcal{G}_1$ and $\mathcal{G}_2$, where the states of $\mathcal{G}_1$ and $\mathcal{G}_2$ are disjoint, then $R_1 \circ R_2$ is recognized by $GTT\_comp(\mathcal{G}_1,\mathcal{G}_2)$.*

The transitive closure of a GTT $\mathcal{G}$ is computed by taking $\mathcal{G}_0 = \mathcal{G}$ and then iterating $\mathcal{G}_{n+1} = GTT\_comp(\mathcal{G}_n,\mathcal{G}_n)$ until a fixed point is reached. If $\mathcal{G}$ is finite, this process terminates. ✔ We have proved that the GTT produced that way accepts the transitive closure of the original GTT. ✔ One interesting aspect is that transitivity of the resulting GTT relation follows immediately from the first part of the proof of Theorem 5 (where the assumption that the states of $\mathcal{G}_1$ and $\mathcal{G}_2$ are disjoint is not used). ✔

Example 6. Returning to our example, let $\mathcal{A}_1 = \mathcal{A} \cup \Delta_\epsilon(\mathcal{B},\mathcal{A})$ and $\mathcal{B}_1 = \mathcal{B} \cup \Delta_\epsilon(\mathcal{A},\mathcal{B})$. The GTT $\mathcal{G}_1 = (\mathcal{A}_1,\mathcal{B}_1)$ recognizes $\to^*_\mathcal{R}$ while its inverse ${}^*_\mathcal{R}\!\leftarrow$ is recognized by $\mathcal{G}_1^- = (\mathcal{B}_1,\mathcal{A}_1)$. Next we compose $\mathcal{G}_1^-$ and $\mathcal{G}_1$ to obtain a GTT $\mathcal{G}_\uparrow$ that recognizes ${}^*_\mathcal{R}\!\leftarrow \cdot \to^*_\mathcal{R}$. This requires a renaming of states in one of the GTTs. Similarly, composing $\mathcal{G}_1$ and $\mathcal{G}_1^-$ produces a GTT $\mathcal{G}_\downarrow$ recognizing the joinability relation $\to^*_\mathcal{R} \cdot {}^*_\mathcal{R}\!\leftarrow$.

Finally, we need to check whether one GTT language is a subset of another one. To this end, we formalized the result ✔ that any GTT relation is an RR_2 relation.

Theorem 7. ✔ *Let $\mathcal{R}$ be a ground TRS and let $\mathcal{G} = (\mathcal{A},\mathcal{B})$ be the GTT simulating $\mathcal{R}$-steps as in Theorem 1. Then $\mathcal{R}$ is confluent on ground terms if and only if*

$$ta\_lang(GTT\_to\_RR2(GTT\_comp(\mathcal{G}^{*-},\mathcal{G}^*))) \subseteq ta\_lang(GTT\_to\_RR2(GTT\_comp(\mathcal{G}^*,\mathcal{G}^{*-})))$$

Example 8. To finish our running example, we transform $\mathcal{G}_\uparrow$ and $\mathcal{G}_\downarrow$ into RR_2 automata. A subsequent language inclusion check returns a negative answer from which we infer that the TRS $\mathcal{R}$ lacks confluence. Indeed there are non-joinable peaks, e.g., $\mathsf{b} \leftarrow \mathsf{a} \to \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{b}) \to \mathsf{c}$.

Note that the results presented so far are purely theoretical, and cannot be executed directly. Here we sketch how to derive executable code for Δ_ϵ, cf. Definition 3. Note that a direct implementation of the definition would require iterating over all ground terms t, of which there are infinitely many. The first step is to define an inductive set Δ'_ϵ ✔ that is equal to Δ_ϵ: ✔

$$\frac{f(\vec{p}) \to p \in \mathcal{A} \quad f(\vec{q}) \to q \in \mathcal{B} \quad \mathsf{len}\,\vec{p} = \mathsf{len}\,\vec{q} = n \quad (p_i,q_i) \in \Delta'_\epsilon\ (1 \leqslant i \leqslant n)}{(p,q) \in \Delta'_\epsilon}\ cong$$

$$\frac{(p,q) \in \Delta'_\epsilon \quad p \to p' \in \mathcal{A}}{(p',q) \in \Delta'_\epsilon}\ \epsilon_1 \qquad \frac{(p,q) \in \Delta'_\epsilon \quad q \to q' \in \mathcal{B}}{(p,q') \in \Delta'_\epsilon}\ \epsilon_2$$

We then plug this into a generic algorithm for Horn inference (which we regard as the foundation of saturation algorithms), which works on inference rules of the shape $a_1 \ \cdots \ a_n \to a$, where a_i, a are all of the same type. The idea here is that proving correctness and termination can be done once and for all on this generic level, and then be reused for any saturation procedure.

In the case of Δ'_ϵ, the inference rules work on pairs of states (p,q), i.e., the potential elements of Δ'_ϵ. We turn the inferences of Δ'_ϵ into Horn clauses by keeping only the premises of the form

$(p, q) \in \Delta_\epsilon'$, evaluating the other premises immediately based on $\mathcal{A}$ and $\mathcal{B}$. ✔ Then we show that the resulting Horn inferences characterize Δ_ϵ'. ✔ In order to use the generic procedure, we have to provide a function that computes the inferences with no premises (Δ_ε'*_infer0*), ✔ and a function that computes inferences that use a particular premise (p, q) and other premises from a given set (Δ_ε'*_infer1*). ✔ With those functions we can instantiate the generic procedure.

$$\Delta_\varepsilon'\_impl\ \mathcal{A}\ \mathcal{B} = saturate\_impl\ (\Delta_\varepsilon'\_infer0\ \mathcal{A}\ \mathcal{B})\ (\Delta_\varepsilon'\_infer1\ \mathcal{A}\ \mathcal{B})$$

Partial correctness follows from partial correctness of the generic procedure. ✔

4 Conclusion

We have outlined an ongoing effort to formalize decidability of (ground) confluence of ground TRSs, which is a useful test case for the decidability of the full first-order theory of rewriting for ground TRSs. The main remaining challenge is to provide executable algorithms for all these results and prove their termination. We have already made significant progress to this end; in fact there are executable versions of all constructions needed for the confluence check, except for the final tree language subset check.

Our immediate goal is to provide a verified confluence checker for ground TRSs. Many tasks remain as future work. We want to adapt the basic TRS to GTT construction to cover the larger class of linear, variable separated (extended) TRSs, which consist of rewrite rules $\ell \to r$ such that ℓ and r are linear terms without common variables. For the full first-order theory of rewriting, while we already have constructions for intersection, union, complement, cylindrification and projection (the latter are used for dealing with quantifiers), these are not yet executable.

References

[1] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available from www.grappa.univ-lille3.fr/tata, 2008.

[2] H. Comon, G. Godoy, and R. Nieuwenhuis. The confluence of ground term rewrite systems is decidable in polynomial time. In *Proc. 42nd FOCS*, pages 298–307, 2001. doi: 10.1109/SFCS.2001.959904.

[3] M. Dauchet, T. Heuillard, P. Lescanne, and S. Tison. Decidability of the confluence of ground term rewriting systems. In *Proc. 2nd LICS*, pages 353–359, 1987.

[4] M. Dauchet, T. Heuillard, P. Lescanne, and S. Tison. Decidability of the confluence of finite ground term rewriting systems and of other related term rewriting systems. *I&C*, 88(2):187–201, 1990. doi: 10.1016/0890-5401(90)90015-A.

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

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

[7] M. Oyamaguchi. The Church-Rosser property for ground term-rewriting systems is decidable. *Theoretical Computer Science*, 49:43–79, 1987. doi: 10.1016/0304-3975(87)90100-9.

[8] F. Rapp and A. Middeldorp. Automating the first-order theory of left-linear right-ground term rewrite systems. In *Proc. 1st FSCD*, volume 52 of *LIPIcs*, pages 36:1–36:12, 2016. doi: 10.4230/LIPIcs.FSCD.2016.36.

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

Convergence of Simultaneously and Sequentially Unraveled TRSs for Normal Conditional TRSs*

Naoki Nishida, Yuta Tsuruta, and Yoshiaki Kanazawa

Nagoya University, Nagoya, Japan
{nishida@, yutsu@trs.css., yoshiaki@trs.css.}i.nagoya-u.ac.jp

Abstract

Unravelings, which are transformations of a conditional term rewriting system (CTRS, for short) into an unconditional term rewriting system (TRS, for short), are useful to prove confluence and operational termination of some CTRSs. A simultaneous unraveling has been proposed for normal 1-CTRSs and a sequential one has been proposed for deterministic 3-CTRSs, the class of which includes normal 1-CTRSs. In this paper, we first show that for a normal 1-CTRS, the simultaneously unraveled TRS is orthogonal iff so is the sequentially unraveled one. Then, we show that for a normal 1-CTRS, if the simultaneously unraveled TRS is terminating, then so is the sequentially unraveled one. Finally, we show that for a normal 1-CTRS with termination of the unraveled TRS, the simultaneously unraveled TRS is locally confluent iff so is the sequentially unraveled one.

1 Introduction

Conditional term rewriting [14, Chapter 7] is known to be much more complicated than unconditional term rewriting in the sense of analyzing properties (cf. [12]). A popular approach to the analysis of conditional term rewriting systems (CTRSs, for short) is to transform a CTRS into an unconditional term rewriting system (TRS, for short) that is in general an overapproximation of the CTRS w.r.t. reduction. This approach enables us to use techniques for the analysis of TRSs, which are well investigated in the literature.

Unravelings [9, 10, 13] are useful to prove *confluence* and *operational termination* [8] of CTRSs because of the following results: (a) a deterministic 3-CTRS (3-DCTRS, for short) is confluent if the unraveled TRS is confluent and the CTRS is *weakly left-linear* (WLL, for short) [6, 7], and (b) a 3-DCTRS is operationally terminating if the unraveled TRS is terminating [3]. A *simultaneous unraveling* has been proposed for normal 1-CTRSs [9, 14], and a *sequential unraveling* has been proposed for 3-DCTRSs [10, 13]. Normal 1-CTRSs are 3-DCTRS and both the simultaneous and sequential unravelings are applicable to normal 1-CTRSs. For this reason, to prove confluence and operational termination of normal 1-CTRSs, we can use both the simultaneous and sequential unravelings. For example, CO3 [11], a confluence prover for CTRSs, tries to prove confluence via the simultaneous unraveling, and, if it fails, then uses the sequential one.

In this paper, we first show that for a normal 1-CTRS, the simultaneously unraveled TRS is orthogonal iff so is the sequentially unraveled one (Section 4). Then, we show that for a normal 1-CTRS, if the simultaneously unraveled TRS is terminating, then so is the sequentially unraveled one (Section 5). Finally, we show that for a normal 1-CTRS with termination of the unraveled TRS, the simultaneously unraveled TRS is locally confluent iff so is the sequentially unraveled one (Section 6). The second and third results imply that for a normal 1-CTRS, the simultaneously unraveled TRS is convergent iff so is the sequentially unraveled one.

*This work was partially supported by JSPS KAKENHI Grant Number JP17H01722.

2 Preliminaries

We omit basic notions and notations for term rewriting [2, 14], and we assume that the reader is familiar with them. In this section, we briefly recall the notions and notations of CTRSs.

An (oriented) *conditional rewrite rule* over a signature $\mathcal{F}$ is a triple (ℓ, r, c), denoted by $\ell \to r \Leftarrow c$, such that the *left-hand side* ℓ is a non-variable term in $T(\mathcal{F}, \mathcal{V})$, the *right-hand side* r is a term in $T(\mathcal{F}, \mathcal{V})$, and the *conditional part* c is a sequence $s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k$ of term pairs ($k \geq 0$) where all of $s_1, t_1, \ldots, s_k, t_k$ are terms in $T(\mathcal{F}, \mathcal{V})$. In particular, a conditional rewrite rule $\ell \to r \Leftarrow c$ is called *unconditional* if the conditional part c is the empty sequence ϵ, and we may abbreviate it to $\ell \to r$. We sometimes attach a unique label ρ to the conditional rewrite rule $\ell \to r \Leftarrow c$ by denoting $\rho : \ell \to r \Leftarrow c$, and we use the label to refer to the rewrite rule. An *(oriented) conditional term rewriting system* (CTRS, for short) over a signature $\mathcal{F}$ is a set of conditional rules over $\mathcal{F}$, and it is called a *term rewriting system* (TRS, for short) if every rule $\ell \to r \Leftarrow c$ in the system is unconditional and $\mathcal{V}ar(\ell) \supseteq \mathcal{V}ar(r)$.

A CTRS $\mathcal{R}$ is called *normal* if for every rule $\ell \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k \in \mathcal{R}$, all of $t_1, \ldots, t_k$ are ground normal forms of $\mathcal{R}_u$ where $\mathcal{R}_u = \{\ell \to r \mid \ell \to r \Leftarrow c \in \mathcal{R}\}$. A CTRS $\mathcal{R}$ is called a *1-CTRS* (*3-CTRS*, resp.) if $\mathcal{V}ar(r, c) \subseteq \mathcal{V}ar(\ell)$ ($\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell, c)$, resp.) for every rule $\ell \to r \Leftarrow c \in \mathcal{R}$. A CTRS $\mathcal{R}$ is called *deterministic* (DCTRS, for short) if for every rule $\ell \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k \in \mathcal{R}$, $\mathcal{V}ar(s_i) \subseteq \mathcal{V}ar(\ell, t_1, \ldots, t_{i-1})$ for all $1 \leq i \leq k$.

3 Unravelings

An *unraveling* U is a transformation of CTRSs into TRSs such that for every CTRS $\mathcal{R}$, we have that $\to_{\mathcal{R}} \subseteq \to^*_{U(\mathcal{R})}$ and $U(\mathcal{R} \cup \mathcal{R}') = U(\mathcal{R}) \cup \mathcal{R}'$ for any TRS $\mathcal{R}'$ [9, 12]. For a CTRS $\mathcal{R}$ over a signature $\mathcal{F}$, we denote the extended signature of $\mathcal{F}$ via U by $\mathcal{F}_{U(\mathcal{R})}$. Given a finite set $X = \{o_1, \ldots, o_n\}$ of objects, a sequence $o_1, o_2, \ldots, o_n$ under some arbitrary order on the objects is denoted by $\overrightarrow{X}$.

A *simultaneous unraveling* for normal 1-CTRSs [9] has been refined as follows.

Definition 3.1 ($\mathbb{U}_{sim}$ [14])**.** *Let $\mathcal{R}$ be a normal 1-CTRS over a signature $\mathcal{F}$. For each conditional rule $\rho : \ell \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k$ in $\mathcal{R}$, we introduce a new function symbol U^ρ, and transform ρ into a set of two unconditional rules as follows:*

$$\mathbb{U}_{sim}(\rho) = \{\ \ \ell \to U^\rho(s_1, \ldots, s_k, \overrightarrow{\mathcal{V}ar(\ell)}),\ \ U^\rho(t_1, \ldots, t_k, \overrightarrow{\mathcal{V}ar(\ell)}) \to r\ \ \}$$

Note that if $k = 0$, then $\mathbb{U}_{sim}(\ell \to r) = \{\ell \to r\}$. $\mathbb{U}_{sim}$ is straightforwardly extended to normal 1-CTRSs: $\mathbb{U}_{sim}(\mathcal{R}) = \bigcup_{\rho \in \mathcal{R}} \mathbb{U}_{sim}(\rho)$. Note that $\mathbb{U}_{sim}(\mathcal{R})$ is a TRS over $\mathcal{F}_{\mathbb{U}_{sim}(\mathcal{R})}$.

A *sequential unraveling* for 3-DCTRSs [10] has been refined as follows.

Definition 3.2 ($\mathbb{U}_{seq}$ [13, 14])**.** *Let $\mathcal{R}$ be a 3-DCTRS over a signature $\mathcal{F}$. For each conditional rule $\rho : \ell \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k$ in $\mathcal{R}$, we introduce k new function symbols $U^\rho_1, \ldots, U^\rho_k$, and transform ρ into a set of $k + 1$ unconditional rules as follows:*

$$\mathbb{U}_{seq}(\rho) = \{\ \ \ell \to U^\rho_1(s_1, \overrightarrow{X_1}),\ \ U^\rho_1(t_1, \overrightarrow{X_1}) \to U^\rho_2(s_2, \overrightarrow{X_2}),\ \ \ldots,\ \ U^\rho_k(t_k, \overrightarrow{X_k}) \to r\ \ \}$$

where $X_i = \mathcal{V}ar(l, t_1, \ldots, t_{i-1})$ for $1 \leq i \leq k$. Note that if $k = 0$, then $\mathbb{U}_{seq}(\ell \to r) = \{\ell \to r\}$. $\mathbb{U}_{seq}$ is straightforwardly extended to DCTRSs: $\mathbb{U}_{seq}(\mathcal{R}) = \bigcup_{\rho \in \mathcal{R}} \mathbb{U}_{seq}(\rho)$. Note that $\mathbb{U}_{seq}(\mathcal{R})$ is a TRS over $\mathcal{F}_{\mathbb{U}_{seq}(\mathcal{R})}$.

Example 3.3. *Consider the following normal 1-CTRS [4] (278.trs in Cops[1]):*

$$\mathcal{R}_1 = \left\{ \begin{array}{l} \mathsf{proc}(y,m) \to \mathsf{proc}(\mathsf{app}(\mathsf{map}(\mathsf{self},\mathsf{nil}),\mathsf{split}_2(m,y)),m) \\ \qquad\qquad \Leftarrow \mathsf{leq}(m,\mathsf{len}(y)) \twoheadrightarrow \mathsf{true}, \mathsf{e}(\mathsf{split}_1(m,y)) \twoheadrightarrow \mathsf{false}, \\ \mathsf{proc}(y,m) \to \mathsf{proc}(\mathsf{split}_2(m,\mathsf{app}(\mathsf{map}(\mathsf{self},\mathsf{nil}),y)),m) \\ \qquad\qquad \Leftarrow \mathsf{leq}(m,\mathsf{len}(y)) \twoheadrightarrow \mathsf{false}, \mathsf{e}(\mathsf{split}_1(m,\mathsf{app}(\mathsf{map}(\mathsf{self},\mathsf{nil}),y))) \twoheadrightarrow \mathsf{false} \end{array} \right\} \cup \mathcal{R}_2$$

where $\mathcal{R}_2$ is a TRS defining app, map, split_2, leq, split_1, *and* e *as in [4, pp. 42–43]. $\mathcal{R}_1$ is unraveled by $\mathbb{U}_{sim}$ and $\mathbb{U}_{seq}$ as follows:*

$$\mathbb{U}_{sim}(\mathcal{R}_1) = \left\{ \begin{array}{l} \mathsf{proc}(y,m) \to U_1(\mathsf{leq}(m,\mathsf{len}(y)),\mathsf{e}(\mathsf{split}_1(m,y)),y,m), \\ U_1(\mathsf{true},\mathsf{false},y,m) \to \mathsf{proc}(\mathsf{app}(\mathsf{map}(\mathsf{self},\mathsf{nil}),\mathsf{split}_2(m,y)),m), \\ \mathsf{proc}(y,m) \to U_2(\mathsf{leq}(m,\mathsf{len}(y)),\mathsf{e}(\mathsf{split}_1(m,\mathsf{app}(\mathsf{map}(\mathsf{self},\mathsf{nil}),y))),y,m), \\ U_2(\mathsf{false},\mathsf{false},y,m) \to \mathsf{proc}(\mathsf{split}_2(m,\mathsf{app}(\mathsf{map}(\mathsf{self},\mathsf{nil}),y)),m) \end{array} \right\} \cup \mathcal{R}_2$$

$$\mathbb{U}_{seq}(\mathcal{R}_1) = \left\{ \begin{array}{l} \mathsf{proc}(y,m) \to U_3(\mathsf{leq}(m,\mathsf{len}(y)),y,m), \\ U_3(\mathsf{true},y,m) \to U_4(\mathsf{e}(\mathsf{split}_1(m,y)),y,m), \\ U_4(\mathsf{false},y,m) \to \mathsf{proc}(\mathsf{app}(\mathsf{map}(\mathsf{self},\mathsf{nil}),\mathsf{split}_2(m,y)),m), \\ \mathsf{proc}(y,m) \to U_5(\mathsf{leq}(m,\mathsf{len}(y)),y,m), \\ U_5(\mathsf{false},y,m) \to U_6(\mathsf{e}(\mathsf{split}_1(m,\mathsf{app}(\mathsf{map}(\mathsf{self},\mathsf{nil}),y))),y,m), \\ U_6(\mathsf{false},y,m) \to \mathsf{proc}(\mathsf{split}_2(m,\mathsf{app}(\mathsf{map}(\mathsf{self},\mathsf{nil}),y)),m) \end{array} \right\} \cup \mathcal{R}_2$$

4 Orthogonality of Unraveled TRSs

In this section, we show that for a normal 1-CTRS $\mathcal{R}$, $\mathbb{U}_{sim}(\mathcal{R})$ is orthogonal (i.e., left-linear and non-overlapping) iff so is $\mathbb{U}_{seq}(\mathcal{R})$.

Let $\mathcal{R}$ be a normal 1-CTRS over a signature $\mathcal{F}$. By definition, for a rule $\ell \to r \in \mathbb{U}_{sim}(\mathcal{R})$, the left-hand side ℓ is either in $T(\mathcal{F},\mathcal{V})$ or of the form $U^\rho(t_1,\ldots,t_k,x_1,\ldots,x_n)$ where $t_1,\ldots,t_k$ are ground normal forms of $\mathcal{R}_u$. In the latter case, the rule $\ell \to r$ is not overlapping with any rule in $\mathbb{U}_{sim}(\mathcal{R})$. For this reason, by definition, if we have two overlapping rules $\ell_1 \to r_1, \ell_2 \to r_2 \in \mathbb{U}_{sim}(\mathcal{R})$, then we have two overlapping rules $\ell_1 \to r_1', \ell_2 \to r_2' \in \mathbb{U}_{seq}(\mathcal{R})$.

Lemma 4.1. *For a normal 1-CTRS $\mathcal{R}$, $\mathbb{U}_{sim}(\mathcal{R})$ is non-overlapping iff so is $\mathbb{U}_{seq}(\mathcal{R})$.*

It follows from the definition of $\mathbb{U}_{sim}$ and [12, Theorem 3.9 (1)] that $\mathbb{U}_{sim}(\mathcal{R})$ is left-linear iff so is $\mathbb{U}_{seq}(\mathcal{R})$. Therefore, the following theorem is a direct consequence of Lemma 4.1.

Theorem 4.2. *For a normal 1-CTRS $\mathcal{R}$, $\mathbb{U}_{sim}(\mathcal{R})$ is orthogonal iff so is $\mathbb{U}_{seq}(\mathcal{R})$.*

Since orthogonality is decidable, given a normal 1-CTRS $\mathcal{R}$, if we prove confluence of an unraveled TRS ($\mathbb{U}_{sim}(\mathcal{R})$ or $\mathbb{U}_{seq}(\mathcal{R})$) via orthogonality, then we can also prove confluence of the other unraveled TRS.

5 Termination of Unraveled TRSs

In this section, we show that for a normal 1-CTRS $\mathcal{R}$, if $\mathbb{U}_{sim}(\mathcal{R})$ is terminating, then so is $\mathbb{U}_{seq}(\mathcal{R})$.

It is shown in [12] that for a normal 1-CTRS $\mathcal{R}$ over a signature $\mathcal{F}$, there exists a *tree homomorphism* $\phi_{\mathcal{R}}$ such that for all terms $s,t \in T(\mathcal{F}_{\mathbb{U}_{seq}(\mathcal{R})},\mathcal{V})$, if $s \to^*_{\mathbb{U}_{seq}(\mathcal{R})} t$, then

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

Table 1: the result of proving termination of the unraveled TRSs from Cops.

result	AProVE		NaTT		CO3	
	$\mathbb{U}_{sim}(\cdot)$	$\mathbb{U}_{seq}(\cdot)$	$\mathbb{U}_{sim}(\cdot)$	$\mathbb{U}_{seq}(\cdot)$	$\mathbb{U}_{sim}(\cdot)$	$\mathbb{U}_{seq}(\cdot)$
YES	39	39	35	35	24	25
NO	9	9	9	9	—	—
MAYBE	0	0	7	7	27	26
timeout (300 seconds)	3	3	0	0	0	0

$\phi_{\mathcal{R}}(s) \to^*_{\mathbb{U}_{sim}(\mathcal{R})} \phi_{\mathcal{R}}(t)$. The tree homomorphism can be extended for *dependency pairs* [1] so that for all terms $s, t \in T(\mathcal{F}_{\mathbb{U}_{seq}(\mathcal{R})}, \mathcal{V})$, if $s^\sharp \to_{DP(\mathbb{U}_{seq}(\mathcal{R}))} t^\sharp$, then $(\phi_{\mathcal{R}}(s))^\sharp$ $(= \cup \to_{DP(\mathbb{U}_{sim}(\mathcal{R}))})$ $(\phi_{\mathcal{R}}(t))^\sharp$, where $DP(\mathcal{R}')$ denotes the set of dependency pairs of a TRS $\mathcal{R}'$ and $u^\sharp$ denotes the term obtained from u by replacing the root symbol by the corresponding *marked* symbol. This implies that if $s, t \in T(\mathcal{F}, \mathcal{V})$ and $s^\sharp$ $(\to^*_{\mathbb{U}_{seq}(\mathcal{R})} \cdot \to_{DP(\mathbb{U}_{seq}(\mathcal{R}))} \cdot \to^*_{\mathbb{U}_{seq}(\mathcal{R})})^+$ $t^\sharp$, then $(\phi_{\mathcal{R}}(s))^\sharp$ $(\to^*_{\mathbb{U}_{sim}(\mathcal{R})} \cdot \to_{DP(\mathbb{U}_{sim}(\mathcal{R}))} \cdot \to^*_{\mathbb{U}_{sim}(\mathcal{R})})^+$ $(\phi_{\mathcal{R}}(t))^\sharp$. Thus, an infinite *dependency chain* of $\mathbb{U}_{seq}(\mathcal{R})$ can be converted to an infinite dependency chain of $\mathbb{U}_{sim}(\mathcal{R})$.

Theorem 5.1. *For a normal 1-CTRS $\mathcal{R}$, if $\mathbb{U}_{sim}(\mathcal{R})$ is terminating, then so is $\mathbb{U}_{seq}(\mathcal{R})$.*

The converse of Theorem 5.1 does not hold in general. For example, for $\mathcal{R}_2 = \{\ \mathsf{a} \to \mathsf{b} \Leftarrow \mathsf{c} \twoheadrightarrow \mathsf{d}, \mathsf{a} \twoheadrightarrow \mathsf{e}\ \}$, $\mathbb{U}_{seq}(\mathcal{R}_2)$ is terminating but $\mathbb{U}_{sim}(\mathcal{R}_2)$ is not.

Since termination is undecidable, unlike orthogonality, Theorem 5.1 does not imply that (a) if we have proved termination of $\mathbb{U}_{sim}(\mathcal{R})$ using some method, then we could directly prove termination of $\mathbb{U}_{seq}(\mathcal{R})$ using some method that does not rely on Theorem 5.1. It is not easy to prove (a) for all existing methods to prove termination of TRSs. Instead of proving (a), we examined (a) for 51 normal 1-CTRSs in Cops.[1] Our experiments were performed on OS X 10.11.6 equipped with an Intel Core i5 CPU at 2.9 GHz with 8 GB RAM, and we used AProVE [5], NaTT [15], and CO3 [11] as termination provers. Table 1 illustrates the number of benchmarks for each result, and indicates that the results for $\mathbb{U}_{sim}$ and $\mathbb{U}_{seq}$ are almost the same—the methods implemented in CO3 are very simple, and thus, the number of YES for $\mathbb{U}_{sim}$ and $\mathbb{U}_{seq}$ are slightly different.

6 Local Confluence of Unraveled TRSs

In this section, we show that for a normal 1-CTRS $\mathcal{R}$ with termination of the unraveled TRSs $\mathbb{U}_{sim}(\mathcal{R})$ and $\mathbb{U}_{seq}(\mathcal{R})$, if $\mathbb{U}_{sim}(\mathcal{R})$ is locally confluent (i.e., confluent), then so is $\mathbb{U}_{seq}(\mathcal{R})$.

Let $\mathcal{R}$ be a normal 1-CTRS over a signature $\mathcal{F}$. As described in Section 4, every overlap of the unraveled TRS is caused by two rules $\ell_1 \to r_1, \ell_2 \to r_2$ such that $\ell_1, \ell_2 \in T(\mathcal{F}, \mathcal{V})$. The tree homomorphism $\phi_{\mathcal{R}}$ in Section 5 can be used for joinability of critical pairs of $\mathbb{U}_{sim}(\mathcal{R})$ from joinability of $\mathbb{U}_{seq}(\mathcal{R})$, and vice versa.

Theorem 6.1. *Let $\mathcal{R}$ be a normal 1-CTRS such that $\mathbb{U}_{sim}(\mathcal{R})$ is terminating. $\mathbb{U}_{sim}(\mathcal{R})$ is locally confluent iff so is $\mathbb{U}_{seq}(\mathcal{R})$.*

For terminating TRSs, (local) confluence is decidable (see [2, p. 140]). Therefore, given a normal 1-CTRS $\mathcal{R}$, if we prove termination of $\mathbb{U}_{sim}(\mathcal{R})$ or $\mathbb{U}_{seq}(\mathcal{R})$, and if we prove local confluence of an unraveled TRS ($\mathbb{U}_{sim}(\mathcal{R})$ or $\mathbb{U}_{seq}(\mathcal{R})$), then we can also prove local confluence of the other unraveled TRS.

7 Conclusion

In this paper, we showed that for a normal 1-CTRS, (1) the simultaneously unraveled TRS is orthogonal iff so is the sequentially unraveled one, (2) if the simultaneously unraveled TRS is terminating, then so is the sequentially unraveled one, and (3) under termination of the unraveled TRS, the simultaneously unraveled TRS is locally confluent iff so is the sequentially unraveled one. The second and third results imply that for a normal 1-CTRS, the simultaneously unraveled TRS is convergent iff so is the sequentially unraveled one. If $\mathcal{R}$ is WLL and $\mathbb{U}_{sim}(\mathcal{R})$ or $\mathbb{U}_{seq}(\mathcal{R})$ is confluent, then $\mathcal{R}$ is confluent [6, 7]. Therefore, to prove confluence of a WLL normal 1-CTRS by either orthogonality of the unraveled TRS or termination and joinability of critical pairs of the unraveled TRS, there is no difference between the use of $\mathbb{U}_{sim}$ and $\mathbb{U}_{seq}$, except for the power of a termination prover we use (see Table 1).

The sequential unraveling has been improved to preserve confluence of CTRSs as much as possible [7, $\mathbb{U}_{conf}$]. We will adapt the results in this paper to the improved sequential unraveling and then we will consider the efficiency of proving confluence via CO3. In addition, we will compare the simultaneous and sequential unravelings w.r.t. other confluence criteria for TRSs.

References

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

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

[3] F. Durán, S. Lucas, J. Meseguer, C. Marché, and X. Urbain. Proving termination of membership equational programs. In *Proc. PEPM 2004*, pp. 147–158. ACM, 2004.

[4] J. Giesl and T. Arts. Verification of Erlang processes by dependency pairs. *Appl. Algebra Eng. Commun. Comput.*, 12(1/2):39–72, 2001.

[5] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR 2006*, volume 4130 of *Lecture Notes in Computer Science*, pp. 281–286. Springer, 2006.

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

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

[8] S. Lucas, C. Marché, and J. Meseguer. Operational termination of conditional term rewriting systems. *Inf. Process. Lett.*, 95(4):446–453, 2005.

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

[10] M. Marchiori. On deterministic conditional rewriting. Computation Structures Group, Memo 405, MIT Laboratory for Computer Science, 1997.

[11] N. Nishida, T. Kuroda, M. Yanagisawa, and K. Gmeiner. CO3: a COnverter for proving COnfluence of COnditional TRSs. In *Proc. IWC 2015*, page 42, 2015.

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

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

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

[15] A. Yamada, K. Kusakari, and T. Sakabe. Nagoya Termination Tool. In *Proc. RTA-TLCA 2014*, volume 8560 of *Lecture Notes in Computer Science*, pp. 466–475. Springer, 2014.

Complete Axiom System of Cluster Algebra

Kousuke Fukui[1] and Koji Nakazawa[2]

Graduate School of Informatics, Nagoya University, Japan
{fukuin.gospel, knak}@sqlab.jp

Abstract

Top trees with DAG representation can be used to compress huge tree data such as XML documents. However, one tree can be represented by several top trees, so it is necessary to efficiently decide which top trees represent the same tree for higher compression rate.

In this paper, we give a complete axiom system for the equational theory of top trees, called the cluster algebra. In order to prove the completeness, we introduce a reduction system on cluster algebra, and show the strong normalization and the unique normal form property.

1 Introduction

Tree-structured data such as XML documents are widely used in the world. In many cases, such data become huge, and it is necessary to compress them. DAG is one of the most common compression techniques, in which equal subtrees are shared. However, a lot of real XML data have common parts not as subtrees but as intermediate structures, called *clusters*, and hence they cannot be shared as subtrees in DAGs. For example, in Figure 1 (a), this tree has the common structure $b(b[\,])$, which is not a subtree but forms a cluster.

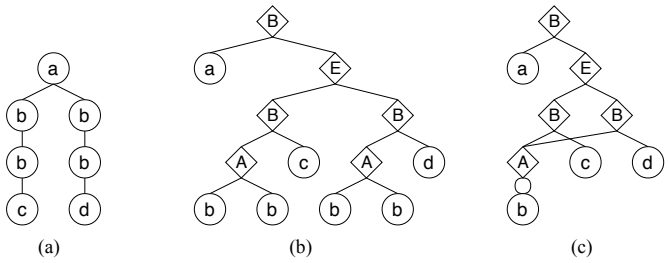


Figure 1: Sharing clusters

To solve this problem, *top trees* and their DAG representations have been proposed [1, 2, 3]. A top tree is a binary tree representing a recipe to reconstructing the original tree by merging its clusters. By the top trees, we can restructure common clusters to subtrees of a top tree, and share them in the top-tree DAGs. In Figure 1 (b), the cluster $b(b[\,])$ is reconstructed as $b\ A\ b$ (we use infix notation for top trees), and it can be shared as a subtree in DAG as (c).

However, the same cluster can be represented by different top trees, and then they cannot be shared in the top-tree DAG. Therefore, if we can efficiently decide whether two top trees represent the same tree, we can expect higher compression rate by the top-tree DAGs.

In this paper, we consider an equational theory for the equivalence of top trees as a theoretical foundation for equivalence checking for top trees. We give an axiom system for the equational theory, called the *cluster algebra* [4], and prove its completeness. For the completeness, we give a reduction system for the cluster algebra, and show the strong normalization and the unique normal form property. We show that there is a one-to-one correspondence between the normal forms and the original trees.

2 Top Tree and Cluster Algebra

We consider ordered trees as original data, in which each of the nodes and leaves has a label, such as $a(b(c,d),e)$. *Clusters* are fragments of a ordered tree. Each cluster has one top boundary node $\perp$ at the root position, and at most one bottom boundary node, which is a distinguished leaf node marked with $[\,]$ such as $a[\,]$. For example, $\perp(b[\,],e)$, $\perp(c,d)$, and $\perp(a(b(c,d),e))$ are clusters in the ordered tree $a(b(c,d),e)$.

We can reconstruct the original ordered tree by merging its clusters. There are five type of merging, which are listed in Figure 2.

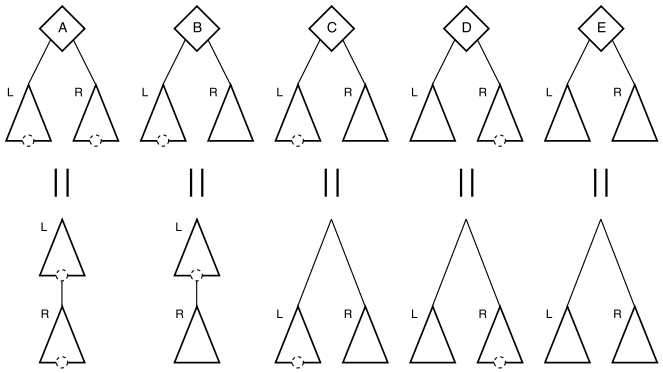


Figure 2: Five types of merging clusters

The type A and B merge two clusters in vertical direction, that is, they replace the bottom boundary node of the left cluster with the right cluster. The difference between A and B is whether the right cluster has a bottom boundary node $[\,]$ or not. The C, D, and E merge two clusters in horizontal direction. For the type C, the left cluster has $[\,]$. For D, the right cluster has $[\,]$. For E, neither has $[\,]$. For example, if two clusters $\perp(b[\,],e)$ and $\perp(c,d)$ are merged by the type B, we obtain the cluster $\perp(b(c,d),e)$. If two clusters $\perp(b,c[\,])$ and $\perp(e)$ are merged by the type C, we obtain

the cluster $\bot(b, c[\,], e)$. We also use the metavariables V for A or B, and H for C, D, or E.

A *top tree* [1, 2] is a binary tree that shows how we can reconstruct the original ordered tree by merging its clusters. Each node of a top tree is one of the merging types A, B, C, D, and E. Each leaf of a top tree is a label of the original ordered tree, which represents either the cluster $\bot(a)$ or $\bot(a[\,])$ depending on the type of the parent node. In [4], the equivalence of the top trees are formalized as the *cluster algebra*, where the merging types are classifying into two kinds, vertical and horizontal merging.

We abuse the metavariables t, $t', \cdots$ for ordered trees and clusters, and we use the metavariables τ, α, $\beta, \cdots$ for top trees. For a cluster t which contains a bottom boundary node and another cluster $t' \equiv \bot(t_1, \cdots, t_n)$, we write $t[t']$ for the cluster obtained by replacing $a[\,]$ in t by $a(t_1, \cdots, t_n)$. For clusters $t = \bot(t_1, \cdots, t_n)$ and $t' = \bot(t'_1, \cdots, t'_m)$, we write $t \odot t'$ for $\bot(t_1, \cdots, t_n, t'_1, \cdots, t'_m)$.

Definition 1. *1. The mapping T from the top trees to the clusters without $[\,]$ and T' from the top trees to the clusters with $[\,]$ are defined as follows.*

$$\begin{aligned} T(a) &= \bot(a) & T'(a) &= \bot(a[\,]) \\ T(\tau_1\ B\ \tau_2) &= T'(\tau_1)[T(\tau_2)] & T'(\tau_1\ A\ \tau_2) &= T'(\tau_1)[T'(\tau_2)] \\ T(\tau_1\ E\ \tau_2) &= T(\tau_1) \odot T(\tau_2) & T'(\tau_1\ C\ \tau_2) &= T'(\tau_1) \odot T(\tau_2) \\ & & T'(\tau_1\ D\ \tau_2) &= T(\tau_1) \odot T'(\tau_2) \end{aligned}$$

The cases which are not listed above are undefined. We call τ well-formed if $T(\tau)$ or $T'(\tau)$ is defined. In the following, we consider only well-formed top trees.

2. When either $T(\tau_1) \equiv T(\tau_2)$ or $T'(\tau_1) \equiv T'(\tau_2)$ holds, τ_1 and τ_2 are said to be equivalent, and we write $\models \tau_1 = \tau_2$.

3 Axioms for Cluster Algebra

We give a set of axioms for equational theory of the cluster algebra.

Definition 2. *1. The axioms for cluster algebra are given as follows.*

$$\begin{aligned} (\alpha\ C\ \beta)\ B\ \gamma &= (\alpha\ B\ \gamma)\ E\ \beta & (\alpha\ E\ \beta)\ E\ \gamma &= \alpha\ E\ (\beta\ E\ \gamma) \\ (\alpha\ C\ \beta)\ A\ \gamma &= (\alpha\ A\ \gamma)\ C\ \beta & (\alpha\ C\ \beta)\ C\ \gamma &= \alpha\ C\ (\beta\ E\ \gamma) \\ (\alpha\ D\ \beta)\ B\ \gamma &= \alpha\ E\ (\beta\ B\ \gamma) & (\alpha\ D\ \beta)\ C\ \gamma &= \alpha\ D\ (\beta\ C\ \gamma) \\ (\alpha\ D\ \beta)\ A\ \gamma &= \alpha\ D\ (\beta\ A\ \gamma) & (\alpha\ E\ \beta)\ D\ \gamma &= \alpha\ D\ (\beta\ D\ \gamma) \\ & & (\alpha\ A\ \beta)\ B\ \gamma &= \alpha\ B\ (\beta\ B\ \gamma) \\ & & (\alpha\ A\ \beta)\ A\ \gamma &= \alpha\ A\ (\beta\ A\ \gamma) \end{aligned}$$

2. We write $\vdash \tau_1 = \tau_2$ if it is derivable by the following inference rules, where $X \in \{A, B, C, D, E\}$.

$$\frac{\tau_1 = \tau_2 \text{ is an axiom}}{\vdash \tau_1 = \tau_2}\ (\textsc{Ax}) \qquad \frac{}{\vdash \tau = \tau}\ (\textsc{Ref}) \qquad \frac{\vdash \tau_1 = \tau_2}{\vdash \tau_2 = \tau_1}\ (\textsc{Sym})$$

$$\frac{\vdash \tau_1 = \tau_2 \quad \vdash \tau_2 = \tau_3}{\vdash \tau_1 = \tau_3}\ (\textsc{Tr}) \qquad \frac{\vdash \tau_1 = \tau_2}{\vdash \tau_1 X \tau = \tau_2 X \tau}\ (\textsc{ComL}) \qquad \frac{\vdash \tau_1 = \tau_2}{\vdash \tau X \tau_1 = \tau X \tau_2}\ (\textsc{ComR})$$

The axioms in the left column exchange V and H. The axioms in the right column are associativity for V and H, respectively.

The soundness is proved by the induction on $\vdash \tau_1 = \tau_2$ straightforwardly.

Theorem 1 (Soundness)**.** *For top trees τ_1 and τ_2, if $\vdash \tau_1 = \tau_2$, then $\models \tau_1 = \tau_2$*

4 Completeness

For the completeness, we introduce a reduction system and prove the strong normalization and the unique normal form property, where we use the fact that there is a one-to-one correspondence between the normal forms and the ordered trees.

Definition 3. *The reduction rules for the cluster algebra are obtained from the axioms in Definition 2.1 by orienting from left to right, such as* $(\ \alpha\ C\ \beta\ )\ B\ \gamma\ \Rightarrow\ (\ \alpha\ B\ \gamma\ )\ E\ \beta$

Theorem 2. *The reduction system for the cluster algebra is strongly normalizable.*

Proof. We define the following three measures.

$$\begin{aligned} w(\tau) &= \Sigma_{V \in \tau}(\text{the number of } H \text{ in the left subtree of } V) \\ d_V(\tau) &= \Sigma_{V \in \tau}(\text{the number of } V \text{ in the left subtree of } V) \\ d_H(\tau) &= \Sigma_{H \in \tau}(\text{the number of } H \text{ in the left subtree of } V) \end{aligned}$$

Then, the pair $(w(\tau), d_V(\tau) + d_H(\tau))$ is strictly decreasing by each reduction step with respect to the lexicographic order. □

The normal forms τ are characterized by the following grammar

$$\tau\ ::= \alpha \mid \alpha\ H\ \tau \qquad\qquad \alpha\ ::= a \mid a\ V\ \tau,$$

Definition 4. *The mapping Θ from the clusters to the normal forms is defined by induction on the size of the clusters as follows.*

$$\begin{aligned} \Theta(\bot(a(t_1, \cdots, t_n))) &= a\ V\ \Theta(\bot(t_1, \cdots, t_n)) \\ \Theta(\bot(t_1, \cdots, t_n)) &= \Theta(\bot(t_1)) H (\cdots (\Theta(\bot(t_{n-1})) H \Theta(\bot(t_n))) \cdots) \end{aligned}$$

Proposition 1. *For any normal form τ, we have $\Theta(T(\tau)) \equiv \tau$.*

The normal forms and the clusters are related as Figure 3.

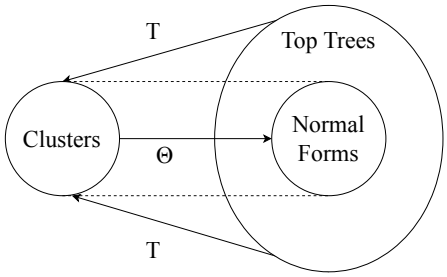


Figure 3: Clusters and normal-form top trees

Proposition 2 (Unique normal form property)**.** *For two normal forms τ_1 and τ_2, if $\models \tau_1 = \tau_2$, then we have $\tau_1 \equiv \tau_2$.*

Proof. By $\models \tau_1 = \tau_2$, we have $T(\tau_1) \equiv T(\tau_2)$, and hence $\Theta(T(\tau_1)) \equiv \Theta(T(\tau_2))$. By the previous proposition, we have $\tau_1 \equiv \tau_2$. □

Theorem 3 (Completeness)**.** *For two top trees τ_1 and τ_2, if $\models \tau_1 = \tau_2$, then $\vdash \tau_1 = \tau_2$.*

Proof. By SN, we have normal forms τ_i' of τ_i for $i = 1, 2$. By the soundness we have $\models \tau_i = \tau_i'$, and by the assumption we have $\models \tau_1' = \tau_2'$. By the previous proposition, we have $\tau_1' \equiv \tau_2'$. Therefore we have $\vdash \tau_1 = \tau_1' \equiv \tau_2' = \tau_2$. □

5 Conclusion

In this paper, we have considered the axioms for the cluster algebra representing the equivalence of the top trees, and proved soundness and completeness of the axiom system. Based on this axiom system, it is expected to be possible to efficiently decide equivalence of top trees without actually decompressing them to the original ordered trees, and higher compression rate in DAG representations of the top trees. As future work, we will give an efficient algorithm for equivalence checker for the top trees.

References

[1] P. Bille, I.L. Gørtz, G.M. Landau, and O. Weimann, Tree compression with top trees, Information and Computation, vol.243, pp.166–177, 2015.

[2] L. Hübschle-Schneider and R. Raman, Tree compression with top trees revisited, Proceedings of the 14th International Symposiumon Experimental Algorithms (SEA2015), LNCS 9125, pp.15–27, 2015.

[3] S. Nishimura, K. Hashimoto, and H. Seki, A Method of Tree Compression with Top Trees and Direct Query Evaluation, IEICE Technical Report 116(127)pp.93–98, 2016. (In Japanese)

[4] A. Gascón, M. Lohrey, S. Maneth, C.P. Reh, and K. Sieber, Grammar-Based Compression of Unranked Trees, Computing Research Repository, `abs/1802.05490`, 2018.

ACP: System Description for CoCo 2018

Takahito Aoto[1] and Yoshihito Toyama[2]

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

A primary functionality of ACP is proving confluence 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 [2]. No new (non-)confluence criterion for TRSs has been incorporated from the one submitted for CoCo 2017.

This year we have added a new functionality to ACP, namely that of proving unique normal forms w.r.t. conversion (UNC) of TRSs. It incorporates divide–and–conquer criteria for UNC and multiple direct criteria for guaranteeing UNC of TRSs. The list of implemented criteria and methods is reported in [3]. In particular, this includes a *UNC completion method* which is inspired from conditional linearlization technique [4], and a UNC criterion of non-duplicating weight-decreasing joinability [5]. A preliminary implementation for proving confluence of (oriented, type 3) conditional term rewriting systems, is also added.

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.ie.riec.tohoku.ac.jp/tools/acp/.

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

[3] T. Aoto and Y. Toyama. Automated proofs of unique normal forms w.r.t. conversion for term rewriting systems. Submitted, 2018.

[4] R.C. de Vrijer. Conditional linearization. *Indagationes Mathematicae*, Vol. 10, No. 1, pp. 145–159, 1999.

[5] Y. Toyama and M. Oyamaguchi. Conditional linearization of non-duplicating term rewriting systems. *IEICE Transactions on Information and Systems*, Vol. E84-D, No. 4, pp. 439–447, 2001.

AGCP: System Description for CoCo 2018

Takahito Aoto[1] and Yoshihito Toyama[2]

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

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

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

No new ground (non-)confluence criterion has been incorporated from the one submitted for CoCo 2017.

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 2018 Participant: CeTA 2.33*

Bertram Felgenhauer, Franziska Rapp, Christian Sternagel, and Sarah Winkler

Department of Computer Science, University of Innsbruck, Austria

The tool CeTA [10] is a certifier for confluence and non-confluence proofs of term rewrite systems (TRSs) and conditional term rewrite systems (CTRSs). Its soundness is proven as part of IsaFoR, the *Isabelle Formalization of Rewriting*. The following techniques are currently supported in CeTA—for further details we refer to the certification problem format (CPF) and to the sources of IsaFoR and CeTA (http://cl-informatik.uibk.ac.at/ceta/).

Term rewrite systems. For terminating systems CeTA can check confluence via the critical pair lemma. For possibly non-terminating TRSs CeTA supports several criteria based on linearity and restricted joinability of critical pairs [5], the rule labeling heuristic [4], addition and removal of redundant rules [3], and terminating critical-pair-closing systems [6]. To certify non-confluence one can provide a divergence and a non-joinability certificate, based on distinct normal forms, *tcap*, interpretations, or tree automata [2]. Since 2018, CeTA features paersistent decomposition [1].

Conditional term rewrite systems. For CTRSs CeTA supports: certifying confluence of almost orthogonal, properly oriented, right-stable 3-CTRSs [7]; unraveling, a technique for transforming a given CTRS into a TRS; confluence of quasi-decreasing strongly deterministic CTRSs, possibly in conjunction with *inlining* [8].

Completion. Since version 2.33 CeTA supports the certification of ordered completion [9].

References

[1] B. Felgenhauer and F. Rapp. Layer systems for confluence – Formalized, 2018. In preparation.

[2] B. Felgenhauer and R. Thiemann. Reachability, confluence, and termination analysis with state-compatible automata. *I&C*, 2016. Available Online.

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

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

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

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

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

[8] C. Sternagel and T. Sternagel. Certifying confluence of quasi-decreasing strongly deterministic-conditional term rewrite systems. In *Proc. 26th CADE*, volume 10395 of *LNCS*, pages 413–431, 2017.

[9] C. Sternagel and S. Winkler. Certified ordered completion. In *Proc. 7th IWC*, 2018.

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

*Supported by Austrian Science Fund (FWF), projects P27502, P27528, and T789.

CO3 (Version 1.5)

Naoki Nishida[1,†], Yuta Tsuruta[1], and Yoshiaki Kanazawa[1,‡]

Nagoya University, Nagoya, Japan

{†nishida@, ‡yoshiaki@trs.css.}i.nagoya-u.ac.jp

CO3, a converter for proving confluence of conditional TRSs, tries to prove confluence of conditional term rewriting systems (CTRSs, for short) by using a transformational approach. The tool is based on the result in [3, 8, 6]: the tool first transforms a given weakly-left-linear (WLL, for short) 3-DCTRS into an unconditional term rewriting system (TRS, for short) by using the *SR transformation* $\mathbb{SR}$ [10, 11, 5] or the *unravelings* $\mathbb{U}_N$ [4] and $\mathbb{U}$ [9], and then verifies confluence of the transformed TRS by using the following theorems: (a) a normal 1-CTRS $\mathcal{R}$ is confluent if $\mathcal{R}$ is WLL and $\mathbb{U}_N(\mathcal{R})$ or $\mathbb{U}(\mathcal{R})$ is confluent [1, 2, 3], (b) a 3-DCTRS $\mathcal{R}$ is confluent if $\mathcal{R}$ is WLL and $\mathbb{U}(\mathcal{R})$ is confluent [2, 3], (c) a WLL normal 1-CTRS $\mathcal{R}$ is confluent if $\mathbb{SR}(\mathcal{R})$ is confluent [8], and (d) a WLL and ultra-WLL 3-DCTRS $\mathcal{R}$ is confluent if $\mathbb{SR}(\mathcal{R})$ is confluent [6]. This tool is basically a converter of CTRSs to TRSs and the main expected use of this tool is the collaboration with other tools for proving confluence of TRSs. For this reason, this tool has very simple and lightweight functions to verify properties such as confluence and termination of TRSs. The tool is available from http://www.trs.css.i.nagoya-u.ac.jp/co3/.

Since version 1.4, CO3 does not use $\mathbb{SR}$ because $\mathbb{SR}(\mathcal{R})$ is not confluent due to some auxiliary rules (see [6]), and the latest version does not use $\mathbb{U}_N$ because the power of proving confluence of normal 1-CTRSs by $\mathbb{U}_N$ is empirically weaker than that by $\mathbb{U}$ under the implemented sufficient conditions for confluence (see [7]).

References

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

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

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

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

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

[6] N. Nishida. Notes on confluence of ultra-WLL SDCTRSs via a structure-preserving transformation. In *Proc. IWC 2016*, pp. 60–64, 2016.

[7] N. Nishida, Y. Tsuruta, and Y. Kanazawa. Convergence of simultaneously and sequentially unraveled trss for normal conditional TRSs. In *Proc. IWC 2018*, pp. 57–62, 2018.

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

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

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

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

CoLL-Saigawa: A Joint Confluence Tool*

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 from:

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

The typical usage is: `collsaigawa <file>`. Here the input file is written in the standard WST format. The tool outputs `YES` if confluence of the input TRS is proved, `NO` if non-confluence is shown, and `MAYBE` if the tool does not reach any conclusion.

CoLL-Saigawa is a joint confluence tool of CoLL v1.2 [8] and Saigawa v1.9 [4]. If an input TRS is left-linear, CoLL proves confluence. Otherwise, Saigawa analyzes confluence. CoLL is a confluence tool specialized for left-linear TRSs. It proves confluence by using Hindley's commutation theorem [3] together with the three commutation criteria: Development closeness [2, 9], rule labeling with weight function [10, 1], and Church-Rosser modulo A/C [6]. Saigawa can deal with non-left-linear TRSs. The tool employs the four confluence criteria: The criteria based on critical pair systems [5, Theorem 3] and on extended critical pairs [7, Theorem 2], rule labeling [10], and Church-Rosser modulo AC [6]. Recently, an implementation bug on the last criterion has been reported (see [11]). We are trying to rectify the bug before the competition.

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

References

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

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

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

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

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

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

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

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

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

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

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

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

CoCo 2018 Participant: ConCon 1.5*

Thomas Sternagel[1], Christian Sternagel[2], and Aart Middeldorp[2]

[1] DVT Innsbruck, Austria
[2] Department of Computer Science, University of Innsbruck, Austria

ConCon is a fully automatic confluence checker for *oriented* first-order conditional term rewrite systems (CTRSs). It is written in Scala and available under the LGPL license at

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

For some of its methods ConCon issues calls to the external unconditional confluence and termination checkers CSI and T$_{\text{T}}$T$_2$ as well as the theorem prover Waldmeister. ConCon first tries to simplify rules and remove infeasible rules from the input system, then it employs the following three confluence criteria:

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

(B) an almost orthogonal extended properly oriented right-stable 3-CTRS is confluent [5],

(C) a deterministic 3-CTRS is confluent if its unraveling is left-linear and confluent [6].

In parallel ConCon also tries to show non-confluence using conditional narrowing (and some other heuristics). To make criteria (A) and (B) more useful, ConCon uses a variety of methods to check for infeasibility of conditional critical pairs, ranging from a simple technique based on unification, via symbol transition graph analysis, reachability problem decomposition, the exploitation of certain equalities in the conditions, and tree automata completion to equational reasoning. ConCon can generate certifiable output [3, 4] for most of the implemented methods. A much more extensive description of ConCon can be found in the recent PhD thesis of the first author [2]. ConCon participates in the categories CTRS and CPF-CTRS at CoCo 2018.

References

[1] J. Avenhaus and C. Loría-Sáenz. On Conditional Rewrite Systems with Extra Variables and Deterministic Logic Programs. In *Proc. 5th LPAR*, volume 822 of *LNAI*, pages 215–229, 1994. doi: 10.1007/3-540-58216-9_40.

[2] T. Sternagel. Reliable Confluence Analysis of Conditional Term Rewrite Systems. PhD thesis, University of Innsbruck, 2017.

[3] C. Sternagel and T. Sternagel. Certifying Confluence of Almost Orthogonal CTRSs via Exact Tree Automata Completion. In *Proc. 1st FSCD*, volume 52 of *LIPIcs*, pages 29:1–29:16, 2016. doi: 10.4230/LIPIcs.FSCD.2016.29.

[4] C. Sternagel and T. Sternagel. Certifying Confluence of Quasi-Decreasing Strongly Deterministic Conditional Term Rewrite Systems. In *Proc. 26th CADE*, volume 10395 of *LNCS*, pages 413–431, 2017. doi: 10.1007/978-3-319-63046-5_26.

[5] T. Suzuki, A. Middeldorp, and T. Ida. Level-Confluence of Conditional Rewrite Systems with Extra Variables in Right-hand Sides. In *Proc. 6th RTA*, volume 914 of *LNCS*, pages 179–193, 1995. doi: 10.1007/3-540-59200-8_56.

[6] R. Thiemann and S. Winkler. Formalizing Soundness and Completeness of Unravelings. In *Proc. 10th FroCoS*, volume 9322 of *LNCS*, pages 239–255, 2015. doi: 10.1007/978-3-319-24246-0_15.

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

CoCo 2018 Participant: CSI 1.2.1*

Bertram Felgenhauer[1], Aart Middeldorp[1], Fabian Mitterwallner[1], and Julian Nagele[2]

[1] Department of Computer Science, University of Innsbruck, Austria
[2] Queen Mary University of London, United Kingdom

CSI is a strong automatic tool for (dis)proving confluence 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 [4]. Compared to last year's version, CSI 1.2.1 contains an implementation of the decision procedure for UNC of linear shallow rewrite systems by Zinn and Verma [5]. Furthermore it supports certified output for the persistent decomposition of many-sorted systems [1–3].

CSI participates in the categories CPF-TRS, NFP, TRS, UNC, and UNR of CoCo 2018.

References

[1] T. Aoto, J. Yoshida, and Y. Toyama. Proving confluence of term rewriting systems automatically. In *Proc. 20th International Conference on Rewriting Techniques and Applications*, volume 5595 of *Lecture Notes in Computer Science*, pages 93–102, 2009. doi: 10.1007/978-3-642-02348-4_7.

[2] B. Felgenhauer, A. Middeldorp, H. Zankl, and V. van Oostrom. Layer systems for proving confluence. *ACM Transactions on Computational Logic*, 16(2:14):1–32, 2015. doi: 10.1145/2710017.

[3] B. Felgenhauer and F. Rapp. Layer systems for confluence – Formalized, 2018. In preparation.

[4] 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.

[5] J. Zinn and R.M. Verma. A polynomial algorithm for uniqueness of normal forms of linear shallow term rewrite systems. *Applicable Algebra in Engineering, Communication and Computing*, 21(6):459–485, 2010. doi: 10.1007/s00200-010-0133-1.

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

CoCo 2018 Participant: CSIˆho 0.3.2*

Julian Nagele

School of Electronic Engineering and Computer Science, Queen Mary University of London, UK
j.nagele@qmul.ac.uk

CSIˆho is a tool for automatically proving confluence of higher-order rewrite systems, specifically pattern rewrite systems (PRSs) as introduced by Nipkow [3, 7]. CSIˆho focuses on patterns in order to ensure decidability of unification for computing critical pairs. To this end CSIˆho implements a version of Nipkow's algorithm for higher-order pattern unification [8]. CSIˆho is an extension of CSI, a powerful confluence prover for first-order term rewrite systems. The tool and a web interface are available at

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

Below we briefly describe the criteria implemented by CSIˆho, a more detailed description of both CSIˆho and CSI can be found in [5, 6].

For terminating PRSs CSIˆho decides confluence by checking joinability of critical pairs [7]. As termination criteria CSIˆho implements a basic higher-order recursive path ordering and static dependency pairs with dependency graph decomposition and the subterm criterion. Alternatively, one can also use an external termination tool like WANDA [2] as an oracle. For potentially non-terminating systems CSIˆho supports weak orthogonality [10] and van Oostrom's result on development closed critical pairs [9]. As a divide-and-conquer technique CSIˆho implements modularity, i.e., decomposing a PRS into parts with disjoint signatures, for left-linear PRSs—note that confluence of PRSs is not modular in general [1]. Moreover CSIˆho uses the simple technique of adding and removing redundant rules [4], adapted for PRSs. New in version 0.3.2 is improved support for showing non-confluence.

References

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

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

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

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

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

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

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

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

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

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

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

CoCo 2018 Participant: FORT 2.0*

Franziska Rapp[1] and Aart Middeldorp[2]

[1] Allgemeines Rechenzentrum Innsbruck, Austria
[2] Department of Computer Science, University of Innsbruck, Austria
aart.middeldorp@uibk.ac.at

FORT is a decision and synthesis tool for the first-order theory of rewriting for finite left-linear right-ground rewrite systems. It implements the decision procedure for this theory, which uses tree automata techniques and goes back to Dauchet and Tison [1]. In this theory confluence-related properties on ground terms are easily expressible. The basic functionality of FORT is described in [2] and in [3] we report on several extensions, including witness generation for existentially quantified variables in formulas and support for combinations of rewrite systems. The latter allows to express a property like commutation, which is a natural generalization of confluence and a potential future CoCo category.

FORT 2.0 is implemented in Java. A command-line version of the tool can be downloaded from

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

FORT participates in the categories GCR, NFP, UNC, and UNR at CoCo 2018.

References

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

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

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

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

The System SOL version 2018

Makoto Hamana[1] and Kentaro Kikuchi[2]

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

SOL is a Haskell-based tool that assists the proofs of confluence and strong normalisation of higher-order computation. SOL is intended to be a generic higher-order computation analysis tool that is applicable to the modern theories of higher-order programming languages. This aim is demonstrated in [Ham17] and further developed in [Ham18].

Based on the foundation of second-order algebraic theories [FH10] and its computational counter part [Ham16, Ham17] and polymorphic extension [Ham18], we implemented various results on higher-order syntax and computation in SOL, including Knuth and Bendix's critical pair checking for confluence, and Function-as-Constructor Unification (FCU) [LM16] for unification. Termination analysis is based on the General Schema criterion [Bla00].

References

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

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

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

[Ham17] M. Hamana. How to prove your calculus is decidable: practical applications of second-order algebraic theories and computation. *Proceedings of the ACM on Programming Languages*, 1(1):22:1–22:28, 9 2017. Vol. Í, Issue ICFP, September 2017, Article No. 22, pp.1-28, ACM Press 2017.

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

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

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