Fundamenta Informaticae 192(2) : 167–218 (2024)
Available at IOS Press through:
`https://doi.org/10.3233/FI-242192`

Proving Confluence in the Confluence Framework with CONFident

Raúl Gutiérrez
DLSIIS, Universidad Politécnica de Madrid, Spain
r.gutierrez@upm.es

Salvador Lucas[*]
DSIC & VRAIN, Universitat Politècnica de València, Spain
slucas@dsic.upv.es

Miguel Vítores
VRAIN, Universitat Politècnica de València, Spain
miguelvitoresv@gmail.com

Abstract. This article describes the *confluence framework*, a novel framework for proving and disproving confluence using a divide-and-conquer modular strategy, and its implementation in CONFident. Using this approach, we are able to automatically prove and disprove confluence of *Generalized Term Rewriting Systems*, where (i) only selected arguments of function symbols can be rewritten and (ii) a rather general class of conditional rules can be used. This includes, as particular cases, several variants of rewrite systems such as (context-sensitive) *term rewriting systems*, *string rewriting systems*, and (context-sensitive) *conditional term rewriting systems*. The divide-and-conquer modular strategy allows us to combine in a proof tree different techniques for proving confluence, including modular decompositions, checking joinability of (conditional) critical and variable pairs, transformations, etc., and auxiliary tasks required by them, e.g., joinability of terms, joinability of conditional pairs, etc.

Keywords: confluence, program analysis, rewriting.

[*]Address for correspondence: Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain.

1. Introduction

Reduction relations $\to$ are pervasive in computer science and semantics of programming languages as suitable means to describe computations $s \to^* t$, where $\to^*$ denotes zero or more steps issued with $\to$. In general, s and t are abstract values (i.e., elements of an arbitrary set A), but often denote program expressions: terms, lambda expressions, configurations in imperative programming, etc. If $s \to^* t$ holds, we say that s *reduces* to t or that t is a *reduct* of s. Confluence is the property of reduction relations guaranteeing that whenever s has two different reducts t and t' (i.e., $s \to^* t$ and $s \to^* t'$), both t and t' are *joinable*, i.e., they have a common reduct u (hence $t \to^* u$ and $t' \to^* u$ holds for some u). Confluence is one of the most important properties of reduction relations: for instance, (i) it ensures that for all expressions s, *at most* one irreducible reduct t of s can be obtained (if any); and (ii) it ensures that two divergent computations can always join in the future. Thus, the semantics and implementation of rewriting-based languages is less dependent on specific strategies to implement reductions.

In this paper, rewriting steps are specified by using *Generalized Term Rewriting Systems*, GTRSs $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ [1], where (i) $\mathcal{F}$ is a signature of function symbols; (ii) Π is a signature of predicate symbols; (iii) replacement restrictions on selected arguments i of k-ary function symbols $f \in \mathcal{F}$ can be specified by means of a *replacement map* μ as in *context-sensitive rewriting* [2]; also, (iv) *atomic* conditions A can be included in the conditional part c of conditional rules $\ell \to r \Leftarrow c$ in R, provided that (v) predicates P occurring in such atomic conditions $P(t_1, \ldots, t_n)$ for terms $t_1, \ldots, t_n$ are *defined* by means of Horn clauses in H.

Since Term Rewriting Systems (TRSs [3]), Conditional TRSs [4], Context-Sensitive TRSs (CS-TRSs [2]), and Context-Sensitive CTRSs [5, Section 8.1] are particular cases of GTRSs (see [1, Section 7.3]), our results apply to all of them.

Example 1.1. Consider the CTRS $\mathcal{R}$ in [6, Example 10] (COPS/387.trs[1]), over a signature $\mathcal{F}$ consisting of function symbols f, g, and s, with R consisting of the *conditional rules*

$$\mathsf{g}(\mathsf{s}(x)) \to \mathsf{g}(x) \tag{1}$$

$$\mathsf{f}(\mathsf{g}(x)) \to x \Leftarrow x \approx \mathsf{s}(\mathsf{0}) \tag{2}$$

where predicate $\approx$ represents a *reachability test* which is performed as part of the preparation of a rewriting step using rule (2). This qualifies $\mathcal{R}$ as an *oriented* CTRS, see, e.g., [7, Definition 7.1.3]. This system is *not* confluent, as we have the following (local) *peak*[2]:

$$\mathsf{f}(\mathsf{g}(\mathsf{0})) \;{}_{(1)}\!\leftarrow \underrightarrow{\mathsf{f}(\overleftarrow{\mathsf{g}(\mathsf{s}(\mathsf{0}))})} \to_{(2)} \mathsf{s}(\mathsf{0}) \tag{3}$$

but $\mathsf{s}(\mathsf{0}))$ and $\mathsf{f}(\mathsf{g}(\mathsf{0}))$ are *irreducible* (no rule applies on them) and hence *not* joinable. Now consider the *replacement map* $\mu_\perp(f) = \emptyset$ for all symbols f, which forbids reductions on *all* arguments of function symbols. Then, we obtain a CS-CTRSs $\mathcal{R}_\perp$, where (3) is *not* possible as the leftmost reduction is forbidden due to $\mu_\perp(\mathsf{f}) = \emptyset$, which disables the rewriting step on $\mathsf{g}(\mathsf{s}(\mathsf{0}))$. Using our framework we are able to prove that $\mathcal{R}$ *is not confluent*, and also that $\mathcal{R}_\perp$ *is confluent*.

[1] *Confluence Problems Data Base*: `http://cops.uibk.ac.at/`.

[2] We use $\underrightarrow{\text{under}}$ and $\overleftarrow{\text{over}}$ arrows to highlight which parts of the expression are rewritten and how.

Confluence has been investigated for several reduction-based formalisms and systems, see, e.g., [7, Chapters 4, 7.3, 7.4 & 8] and the references therein. Confluence is *undecidable* already for TRSs, see, e.g., [7, Section 4.1]. Since TRSs are GTRSs, it is undecidable for GTRSs too. Thus, no algorithm is able to prove or disprove confluence of the reduction relation associated to all such systems. Hence, existing techniques for proving and disproving confluence are *partial*, i.e., they succeed on some kinds of systems and fail on others. However, the combination of techniques in a certain order or the use of auxiliary properties can help to prove or disprove confluence.

In this paper, we introduce a *confluence framework* for proving confluence of GTRSs, inspired by the *Dependency Pair Framework*, originally developed for proving (innermost) termination of TRSs [8, 9] and then generalized and extended to cope with other kind of termination problems [10, 11, 12, 13, 14], including termination of GTRSs [15], and also to other properties like *feasibility*, which tries to find a substitution satisfying a combination of atoms with respect to a first-order theory [16].

In the confluence framework, we define two kinds of problems: three variants of *confluence problems* and two variants of *joinability problems*. *Confluence problems* encapsulate the system $\mathcal{R}$ whose confluence is tested. Such confluence problems are transformed, decomposed, simplified, etc., into other (possibly different) problems by using the so-called *processors* that can be plugged in and out in a proof strategy, allowing us to find the best place to apply a proof technique in practice. Processors embed existing results about confluence of (variants of) rewrite systems, confluence-preserving transformations, etc. Besides, *joinability problems* are produced by some processors acting on confluence problems. They are used to prove or disprove the joinability of conditional pairs (including conditional critical pairs [17, Definition 3.2] and conditional variable pairs [1, Definition 59]). They are also treated by appropriate processors. The obtained proof is depicted as a labeled *proof tree* from which the (non-)confluence of the targetted rewrite system can be proved. Processors apply on the obtained problems until (i) a trivial problem is obtained (which is then labeled with yes) and the proof either continues by considering pending problems, or else *finishes* and yes is returned if no problem remains to be solved; or (ii) a counterexample is obtained and the problem is then labeled with no and the proof *finishes* as well but no is returned; or (iii) the successive application of all available processors finishes *unsuccessfully* and then the whole proof finishes unsuccessfully (and 'MAYBE' is returned); or (iv) the ongoing proof is eventually interrupted due to a *timeout*, which is usually prescribed in this kind of proof processes whose termination is not guaranteed or could take too much time, and the whole proof fails. The use of processors often requires calls to external tools to solve proof obligations like *termination*, *feasibility*, *theorem proving*, etc.

This paper is an extended and revised version of [18]. The main differences are:

1. The confluence framework has been extended to cope with Generalized Term Rewriting Systems, thus extending the scope of [18], where only TRSs, CS-TRSs, and CTRSs were treated. In particular, we can treat CS-CTRSs now as particular cases of GTRSs.

2. In [18] only *confluence* and *joinability* problems were considered. Additional related problems are considered now: *local* and *strong* confluence problems, and *strong joinability* problems. They permit a better organization of confluence proofs. Of course, they also permit the use of the framework for (dis)proving local confluence and strong confluence of GTRSs.

3. 16 processors applicable to GTRSs are described in this paper (versus 10 in [18]). They apply on (local, strong) confluence problems for GTRSs.
4. Details about the *implementation* of the confluence framework in CONFident are given now, including a more precise description of CONFident proof strategy and its implementation.
5. Updated information about the participation of CONFident in the 2023 International Confluence Competition, CoCo 2023, is provided.

After some preliminaries in Section 2, Section 3 describes Generalized Term Rewriting Systems. Section 4 defines the problems and processors used in the confluence framework. Section 5 gives a list of processors that can be used in the framework. Section 6 presents the proof strategy of CONFident. Section 7 provides some details about the general implementation of CONFident. Section 8 provides an experimental evaluation of the tool, including an analysis of the use of processors in proofs of (non-)confluence. Section 9 discusses related work. Section 10 concludes.

2. Preliminaries

In the following, *w.r.t.* means *with respect to* and *iff* means *if and only if*. We assume some familiarity with the basic notions of term rewriting [3, 7, 19] and first-order logic [20, 21], where missing definitions can be found. For the sake of readability, though, here we summarize the main notions and notations we use.

Abstract Reduction Relations. Given a binary relation $\mathsf{R} \subseteq A \times A$ on a set A, we often write $a\ \mathsf{R}\ b$ or $b\ \mathsf{R}^{-1}\ a$ instead of $(a,b) \in \mathsf{R}$. The *reflexive* closure of R is denoted by $\mathsf{R}^{=}$; the *transitive* closure of R is denoted by R^{+}; and the *reflexive and transitive* closure by R^{*}. Given $a \in A$, and a relation R, let $\mathsf{Suc}_{\mathsf{R}}(a) = \{b \mid a\ \mathsf{R}^*\ b\}$ be the set of R-*successors* of a (and then adding a itself) and $\mathsf{Suc}^{=}_{\mathsf{R}}(a) = \{b \mid a\ \mathsf{R}^{=}\ b\}$ be the set of *direct* R-*successors* of a (and also adding a). An element $a \in A$ is *irreducible*, if there is no b such that $a\ \mathsf{R}\ b$; we say that b is an R-normal form of a (written $a\ \mathsf{R}^!\ b$), if $a\ \mathsf{R}^*\ b$ and b is an R-normal form. We say that $b \in A$ is R-reachable from $a \in A$ if $a\ \mathsf{R}^*\ b$. We say that $a, b \in A$ are R-*joinable* if there is $c \in A$ such that $a\ \mathsf{R}^*\ c$ and $b\ \mathsf{R}^*\ c$. We say that $a, b \in A$ are *strongly* R-*joinable* if there are $c, c' \in A$ such that $a\ \mathsf{R}^{=}\ c$, $b\ \mathsf{R}^*\ c$, $a\ \mathsf{R}^*\ c'$, and $b\ \mathsf{R}^{=}\ c'$. Also, $a, b \in A$ are R-*convertible* if there is $c \in A$ such that $a\ (\mathsf{R} \cup \mathsf{R}^{-1})^* b$. Given $a \in A$, if there is no infinite sequence $a = a_1\ \mathsf{R}\ a_2\ \mathsf{R}\ \cdots\ \mathsf{R}\ a_n\ \mathsf{R} \cdots$, then a is R-*terminating*; R is *terminating* if a is R-terminating for all $a \in A$. We say that R is (locally) *confluent* if, for every $a, b, c \in A$, whenever $a\ \mathsf{R}^*\ b$ and $a\ \mathsf{R}^*\ c$ (resp. $a\ \mathsf{R}\ b$ and $a\ \mathsf{R}\ c$), b and c are R-joinable. Also, R is *strongly confluent* if, for every $a, b, c \in A$, whenever $a\ \mathsf{R}\ b$ and $a\ \mathsf{R}\ c$, b and c are strongly R-joinable.

Signatures, Terms, Positions. In this paper, $\mathcal{X}$ denotes a countable set of *variables*. A *signature of symbols* is a set of *symbols* each with a fixed *arity*. We use $\mathcal{F}$ to denote a *signature of function symbols* $f, g, \ldots$, whose arity is given by a mapping $ar : \mathcal{F} \to \mathbb{N}$. The set of terms built from $\mathcal{F}$ and $\mathcal{X}$ is $\mathcal{T}(\mathcal{F}, \mathcal{X})$. The set of variables occurring in t is $\mathcal{V}ar(t)$; we often use $\mathcal{V}ar(t, t', \ldots)$ to denote the set of variables occurring in a sequence of terms. Terms are viewed as labeled trees in the usual way. *Positions* p are represented by chains of positive natural numbers used to address subterms $t|_p$ of t. The *set of positions* of a term t is $\mathcal{P}os(t)$. The set of positions of a subterm s in t is denoted $\mathcal{P}os_s(t)$. The set of positions of non-variable symbols in t are denoted as $\mathcal{P}os_{\mathcal{F}}(t)$.

Replacement Maps. Given a signature $\mathcal{F}$, a *replacement map* is a mapping μ satisfying that, for all symbols f in $\mathcal{F}$, $\mu(f) \subseteq \{1, \ldots, ar(f)\}$ [2]. The set of replacement maps for the signature $\mathcal{F}$ is $M_{\mathcal{F}}$. Extreme cases are $\mu_\bot$, disallowing replacements in all arguments of function symbols: $\mu_\bot(f) = \emptyset$ for all $f \in \mathcal{F}$; and $\mu_\top$, restricting no replacement: $\mu_\top(f) = \{1, \ldots, k\}$ for all k-ary $f \in \mathcal{F}$. The set $\mathcal{P}os^\mu(t)$ of *μ-replacing (or* active*) positions* of t is $\mathcal{P}os^\mu(t) = \{\Lambda\}$, if $t \in \mathcal{X}$, and $\mathcal{P}os^\mu(t) = \{\Lambda\} \cup \{i.p \mid i \in \mu(f), p \in \mathcal{P}os^\mu(t_i)\}$, if $t = f(t_1, \ldots, t_k)$. The set of *non-μ-replacing* (or *frozen*) positions of t is $\overline{\mathcal{P}os^\mu}(t) = \mathcal{P}os(t) - \mathcal{P}os^\mu(t)$. Positions of *active* non-variable symbols in t are denoted as $\mathcal{P}os^\mu_{\mathcal{F}}(t)$. Given a term t, $\mathcal{V}ar^\mu(t)$ (resp. $\mathcal{V}ar^{\not\mu}(t)$) is the set of variables occurring at active (resp. frozen) positions in t: $\mathcal{V}ar^\mu(t) = \{x \in \mathcal{V}ar(t) \mid \exists p \in \mathcal{P}os^\mu(t), x = t|_p\}$ and $\mathcal{V}ar^{\not\mu}(t) = \{x \in \mathcal{V}ar(t) \mid \exists p \in \overline{\mathcal{P}os^\mu}(t), x = t|_p\}$. In general, $\mathcal{V}ar^\mu(t)$ and $\mathcal{V}ar^{\not\mu}(t)$ are not disjoint: $x \in \mathcal{V}ar(t)$ may occur active and also frozen in t.

Unification. A renaming ρ is a bijection from $\mathcal{X}$ to $\mathcal{X}$. A substitution σ is a mapping $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ from variables into terms which is homomorphically extended to a mapping (also denoted σ) $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$. It is standard to assume that substitutions σ satisfy $\sigma(x) = x$ except for a *finite* set of variables. Thus, we often write $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where $t_i \neq x_i$ for $1 \leq i \leq n$ to denote a substitution. Terms s and t *unify* if there is a substitution σ (i.e., a *unifier*) such that $\sigma(s) = \sigma(t)$. If s and t unify, then there is a (unique, up to renaming) *most general unifier* (*mgu*) θ of s and t satisfying that, for any other unifier σ of s and t, there is a substitution τ such that, for all $x \in \mathcal{X}$, $\sigma(x) = \tau(\theta(x))$.

First-Order Logic. Here, Π denotes a signature of *predicate symbols*. Atoms and first-order formulas are built using such function and predicate symbols, variables in $\mathcal{X}$, quantifiers $\forall$ and $\exists$ and logical connectives for conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), and implication ($\Rightarrow$), in the usual way. A first-order theory (FO-theory for short) Th is a set of sentences (formulas whose variables are all *quantified*). In the following, given an FO-theory Th and a formula φ, $\mathsf{Th} \vdash \varphi$ means that φ is *deducible* from (or a *logical consequence* of) Th by using a correct and complete deduction procedure.

Feasibility Sequences. An *f-condition* γ is an atom [16]. Sequences $\mathsf{F} = (\gamma_i)_{i=1}^n = (\gamma_1, \ldots, \gamma_n)$ of f-conditions are called *f-sequences*. We often drop 'f-' when no confusion arises. Given an FO-theory Th, a condition γ is Th*-feasible* (or just *feasible* if no confusion arises) if $\mathsf{Th} \vdash \sigma(\gamma)$ holds for some substitution σ; otherwise, it is *infeasible*. A sequence F is Th-feasible (or just *feasible*) if there is a substitution σ satisfying all conditions in the sequence, i.e., for all $\gamma \in \mathsf{F}$, $\mathsf{Th} \vdash \sigma(\gamma)$ holds.

Grounding variables. Let $\mathcal{F}$ be a signature and $\mathcal{X}$ be a set of variables such that $\mathcal{F} \cap \mathcal{X} = \emptyset$. Let $\mathcal{F}_{\mathcal{X}} = \mathcal{F} \cup C_{\mathcal{X}}$ where variables $x \in \mathcal{X}$ are considered as (different) *constant* symbols c_x of

$C_\mathcal{X} = \{c_x \mid x \in \mathcal{X}\}$ and $\mathcal{F}$ and $C_\mathcal{X}$ are disjoint [22], see also [23, page 224]. Given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, a ground term $t^\downarrow$ is obtained by replacing each occurrence of $x \in \mathcal{X}$ in t by c_x. Given a substitution $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, we define $\sigma^\downarrow = \{x_1 \mapsto t_1^\downarrow, \ldots, x_n \mapsto t_n^\downarrow\}$.

3. Generalized term rewriting systems

The material in this section is taken from [1, Section 7]. We consider definite Horn clauses $\alpha : A \Leftarrow c$ (with label α) where c is a sequence $A_1, \ldots, A_n$ of atoms. If $n = 0$, then α is written A rather than $A \Leftarrow$. Let $\mathcal{F}$ be a signature of function symbols, Π be a signature of predicate symbols, $\mu \in M_\mathcal{F}$ be a replacement map, H be a set of clauses $A \Leftarrow c$ where $root(A) \notin \{\to, \to^*\}$, and R be a set of *rewrite rules* $\ell \to r \Leftarrow c$ such that ℓ is not a variable (in both cases, c is a sequence $A_1, \ldots, A_n$ of atoms). The tuple $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ is called a *Generalized Term Rewriting System* (GTRS, [1, Definition 51]). As in [24, Definition 6.1], rules $\ell \to r \Leftarrow c \in R$ are classified according to the distribution of variables: type 1, if $\mathcal{V}ar(r) \cup \mathcal{V}ar(c) \subseteq \mathcal{V}ar(\ell)$; type 2, if $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell)$; type 3, if $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell) \cup \mathcal{V}ar(c)$; and type 4, otherwise. A rule of type n is often called an n-rule. A GTRS $\mathcal{R}$ is called an n-GTRS if all its rules are of type n; if $\mathcal{R}$ contains at least one n-rule which is *not* an m-rule for some $m < n$, then we say that $\mathcal{R}$ is a *proper* n-GTRS. The FO-theory of a GTRS $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ is

$$\overline{\mathcal{R}} = \{(\text{Rf}), (\text{Co})\} \cup \{(\text{Pr})_{f,i} \mid f \in \mathcal{F}, i \in \mu(f)\} \cup \{(\text{HC})_\alpha \mid \alpha \in H \cup R\}$$

where, as displayed in Table 1, (Rf) expresses *reflexivity* of many-step rewriting; (Co) expresses

Table 1. Generic sentences of the first-order theory of rewriting

Label	Sentence
(Rf)	$(\forall x)\, x \to^* x$
(Co)	$(\forall x, y, z)\, x \to y \wedge y \to^* z \Rightarrow x \to^* z$
$(\text{Pr})_{f,i}$	$(\forall x_1, \ldots, x_k, y_i)\, x_i \to y_i \Rightarrow f(x_1, \ldots, x_i, \ldots, x_k) \to f(x_1, \ldots, y_i, \ldots, x_k)$
$(\text{HC})_{A \Leftarrow A_1, \ldots, A_n}$	$(\forall x_1, \ldots, x_p)\, A_1 \wedge \cdots \wedge A_n \Rightarrow A$
	where $x_1, \ldots, x_p$ are the variables occurring in $A_1, \ldots, A_n$ and A

compatibility of one-step and many-step rewriting; for each k-ary function symbol f, $i \in \mu(f)$, and $x_1, \ldots, x_k$ and y_i distinct variables, $(\text{Pr})_{f,i}$ enables the *propagation* of rewriting steps in the i-th immediate *active* subterm of a term with root symbol f; finally, for each Horn clause $\alpha \in H \cup R$, $(\text{HC})_\alpha$ makes explicit the relationship between Horn clause symbol $\Leftarrow$ (also used in rewrite rules which are particular Horn clauses, actually) and logic implication $\Rightarrow$.

Definition 3.1. (Rewriting as deduction)
Let $\mathcal{R}$ be a GTRS. For all terms s and t, we write $s \to_\mathcal{R} t$ (resp. $s \to_\mathcal{R}^* t$) if $\overline{\mathcal{R}} \vdash s \to t$ (resp. $\overline{\mathcal{R}} \vdash s \to^* t$).

Join	(J)	$(\forall x, y, z)$	$x \to^* z \wedge y \to^* z \Rightarrow x \approx y$
Oriented	(O)	$(\forall x, y)$	$x \to^* y \Rightarrow x \approx y$
	(SE_1)	$(\forall x)$	$x \approx x$
Semi-equational	(SE_2)	$(\forall x, y, z)$	$x \to y \wedge y \approx z \Rightarrow x \approx z$
	(SE_3)	$(\forall x, y, z)$	$y \to x \wedge y \approx z \Rightarrow x \approx z$

Figure 1. Sentences for different semantics of CS-CTRSs

Figure 1 displays some sentences which can be included in H to make the *meaning* of predicate $\approx$ often used in the conditions of rules explicit; see, e.g., [7, Definition 7.1.3]: (J) interprets $\approx$ as $\to_{\mathcal{R}}$*-joinability* of terms; (O) interprets $\approx$ as $\to_{\mathcal{R}}$*-reachability*; and $(\mathrm{SE})_1$, $(\mathrm{SE})_2$, $(\mathrm{SE})_3$ provide the interpretation of $\approx$ as $\to_{\mathcal{R}}$*-conversion*. Accordingly, we let $H_\approx = \{(\mathrm{J})\}$, or $H_\approx = \{(\mathrm{O})\}$, or $H_\approx = \{(\mathrm{SE})_1, (\mathrm{SE})_2, (\mathrm{SE})_3\}$ and then we include $H_\approx$ in H. Given a GTRS $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$, a number of well-known classes of rule-based systems is obtained:

- if $\Pi = \{\to, \to^*\}$, $\mu = \mu_\top$, and R consists of unconditional 2-rules only, then $\mathcal{R}$ is a TRS and we often just refer to it as $(\mathcal{F}, R)$.
- if $\Pi = \{\to, \to^*\}$ and R consists of unconditional 2-rules only, then $\mathcal{R}$ is a CS-TRS [2] and we often just refer to it as $(\mathcal{F}, \mu, R)$.
- if $\Pi = \{\to, \to^*, \approx\}$ and for all $\ell \to r \Leftarrow c \in R$, (i) c consists of conditions $s \approx t$ (for some terms s and t), and (ii) $H = H_\approx$ then, depending on $H_\approx$, as explained above, $\mathcal{R}$ is a (J-,O-,SE-)CS-CTRS; if $\mu = \mu_\top$, then $\mathcal{R}$ is a (J-,O-,SE-)CTRS. If no confusion arises, we often use $(\mathcal{F}, \mu, R)$ and $(\mathcal{F}, R)$ instead of $(\mathcal{F}, \Pi, \mu, H_\approx, R)$ or $(\mathcal{F}, \Pi, \mu_\top, H_\approx, R)$, although these notations are more self-contained as H embeds the evaluation semantics of $\approx$ [1, Remark 53].

Definition 3.2. (Confluence and termination of GTRSs)
A GTRS $\mathcal{R}$ is (locally) confluent (resp. terminating) if $\to_{\mathcal{R}}$ is (locally) confluent (resp. terminating).

Remark 3.3. (Confluence and termination of CS-TRSs and CS-CTRSs)
As remarked above, TRSs, CS-TRSs, CTRSs, and CS-CTRSs are particular cases of GTRSs. In the realm of context-sensitive rewriting, it is often useful to make explicit the replacement map μ when referring to the context-sensitive rewriting relation (by writing $\hookrightarrow_{\mathcal{R},\mu}$ and $\hookrightarrow^*_{\mathcal{R},\mu}$, or just $\hookrightarrow$ and $\hookrightarrow^*$) and computational properties of CS-TRSs and CS-CTRSs $\mathcal{R}$ using a replacement map μ, i.e., we usually talk of μ-termination or (local, strong) μ-confluence of $\mathcal{R}$, see [2]. This is useful to *compare* properties of context-sensitive systems and the corresponding properties of unrestricted systems (TRSs and CTRSs). In this paper, though, we use a uniform notation, $\to_{\mathcal{R}}$, for the rewrite relation associated to a GTRS $\mathcal{R}$, and also a uniform designation of the properties, just following Definition 3.2.

A rule $\alpha : \ell \to r \Leftarrow c \in \mathcal{R}$ is *(in)feasible* if c is $\overline{\mathcal{R}}$-(in)feasible. Two terms s and t are $\to_{\mathcal{R}}$-joinable iff $s^\downarrow$ and $t^\downarrow$ are $\to_{\mathcal{R}}$-joinable, cf. [22, Proposition 6]. As in [1, Section 5] we consider *conditional pairs* $\langle s, t\rangle \Leftarrow A_1, \ldots, A_n$, where s, t are terms and $A_1, \ldots, A_n$ are atoms. For a GTRS $\mathcal{R}$, we say that $\pi : \langle s, t\rangle \Leftarrow c$ is *(in)feasible* if c is $\overline{\mathcal{R}}$-(in)feasible. Also, π is (strongly) *joinable* if for all substitutions

σ, whenever $\overline{\mathcal{R}} \vdash \sigma(\gamma)$ holds for all $\gamma \in c$, terms $\sigma(s)$ and $\sigma(t)$ are (strongly) joinable. A conditional pair π is *trivial* if $s = t$. Trivial and infeasible conditional pairs are both joinable.

Definition 3.4. (Extended critical pairs of a GTRS, [1, Definitions 59 & 60])
Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ be a GTRS and $\alpha : \ell \to r \Leftarrow c, \alpha' : \ell' \to r' \Leftarrow c' \in R$ feasible rules sharing no variable (rename if necessary).

- Let $p \in \mathcal{P}os^{\mu}_{\mathcal{F}}(\ell)$ be a nonvariable position of ℓ such that $\ell|_p$ and ℓ' unify with *mgu* θ. Then,
$$\langle \theta(\ell[r']_p), \theta(r) \rangle \Leftarrow \theta(c), \theta(c') \tag{4}$$
is a *conditional critical pair* (CCP) of $\mathcal{R}$. If $p = \Lambda$ and α' is a renamed version of α, then (4) is called *improper*; otherwise, it is called *proper*.
- Let $x \in \mathcal{V}ar^{\mu}(\ell), p \in \mathcal{P}os^{\mu}_{x}(\ell)$, and x' be a fresh variable. Then,
$$\langle \ell[x']_p, r \rangle \Leftarrow x \to x', c \tag{5}$$
is a *conditional variable pair* (CVP) of $\mathcal{R}$. Variable x is called the *critical variable* of the pair.

In both cases, p is called the *critical position*. We use the following notation:

- $\mathsf{pCCP}(\mathcal{R})$ is the set of feasible *proper conditional critical pairs* of $\mathcal{R}$;
- $\mathsf{iCCP}(\mathcal{R})$ is the set of feasible *improper conditional critical pairs* of 3-rules in $\mathcal{R}$ (as improper critical pairs of 2-rules are joinable); and
- $\mathsf{CVP}(\mathcal{R})$ is the set of all *feasible conditional variable pairs* in $\mathcal{R}$.

Then,
$$\mathsf{ECCP}(\mathcal{R}) = \mathsf{pCCP}(\mathcal{R}) \cup \mathsf{iCCP}(\mathcal{R}) \cup \mathsf{CVP}(\mathcal{R})$$
is the set of *extended conditional critical pairs* of $\mathcal{R}$.

For unconditional systems (CS-TRSs and TRSs) we can focus on smaller sets of possibly conditional pairs to analize confluence:

- For CS-TRSs $\mathcal{R}$, we have $\mathsf{iCCP}(\mathcal{R}) = \emptyset$ ($\mathcal{R}$ contains no 3-rule), and $\mathsf{pCCP}(\mathcal{R})$ is written $\mathsf{CP}(\mathcal{R}, \mu)$ or just $\mathsf{CP}(\mathcal{R})$ if no confusion arises. Also, following [25, Definition 20], instead of $\mathsf{CVP}(\mathcal{R})$, we can use the set of LH_μ-critical pairs
$$\begin{aligned} \mathsf{LHCP}(\mathcal{R}, \mu) &= \{\langle \ell[x']_p, r \rangle \Leftarrow x \to x' \mid \ell \to r \in \mathcal{R}, x \in \mathcal{V}ar^{\mu}(\ell) \cap (\mathcal{V}ar^{\not\mu}(\ell) \cup \mathcal{V}ar^{\not\mu}(r)), \\ &\qquad p \in \mathcal{P}os^{\mu}_{x}(\ell)\} \\ &\subseteq \mathsf{CVP}(\mathcal{R}) \end{aligned}$$
which provides a more specific set of conditional pairs obtained from an unconditional rule $\ell \to r$ capturing *possibly harmful* peaks coming from variables which are *active* in the left-hand side ℓ but are also *frozen* in the same ℓ or else in the right-hand side r of the rule (see [1, Section 8] for a comparison of LH_μ-critical pairs and conditional variable pairs for a CS-TRS $\mathcal{R}$).
- For TRSs $\mathcal{R}$ (where $\mu_\top$ can be assumed to view it as a CS-TRS), $\mathsf{iCCP}(\mathcal{R}) = \emptyset$ (no 3-rules), $\mathsf{LHCP}(\mathcal{R}, \mu_\top) = \emptyset$ (as $\mathcal{V}ar^{\not\mu_\top}(t) = \emptyset$ for all terms t) and $\mathsf{pCCP}(\mathcal{R})$ is written $\mathsf{CP}(\mathcal{R})$.

Example 3.5. Consider the CTRS $\mathcal{R}$ in Example 1.1. With rule (2), i.e., $\mathsf{f}(\mathsf{g}(x)) \to x \Leftarrow x \approx \mathsf{s}(\mathsf{0})$, $1 \in \mathcal{P}os_{\mathcal{F}}(\ell_{(2)})$ and (1)', i.e., $\mathsf{g}(\mathsf{s}(x')) \to \mathsf{g}(x')$, since $\mathsf{f}(\mathsf{g}(x))|_1 = \mathsf{g}(x)$ and $\mathsf{g}(\mathsf{s}(x'))$ unify with $\theta = \{x \mapsto \mathsf{s}(x')\}$, we obtain the following (feasible) *proper conditional critical pair*

$$\langle \mathsf{f}(\mathsf{g}(x')), \mathsf{s}(x') \rangle \Leftarrow \mathsf{s}(x') \approx \mathsf{s}(\mathsf{0}) \tag{6}$$

Thus, $\mathsf{pCCP}(\mathcal{R}) = \{(6)\}$. Since $\mathcal{R}$ is a 1-CTRS, $\mathsf{iCCP}(\mathcal{R}) = \emptyset$. With (2) and $x \in \mathcal{V}ar^{\mu_\top}(\mathsf{f}(\mathsf{g}(x))) = \mathcal{V}ar(\mathsf{f}(\mathsf{g}(x)))$, we obtain $\mathsf{CVP}(\mathcal{R}) = \{(7)\}$ for the following *conditional variable pair*:

$$\langle \mathsf{f}(\mathsf{g}(x')), x \rangle \Leftarrow x \to x', x \approx \mathsf{s}(\mathsf{0}) \tag{7}$$

Thus, $\mathsf{ECCP}(\mathcal{R}) = \{(6), (7)\}$.

Example 3.6. For the CS-CTRS $\mathcal{R}_\bot$ in Example 1.1 there is no conditional critical pair because the only active position of the left-hand sides $\ell_{(1)}$ and $\ell_{(2)}$ of rules (1) and (2) is Λ. However, $\ell_{(1)}$ and $\ell_{(2)}$ do *not* unify. Hence $\mathsf{pCCP}(\mathcal{R}_\bot) = \emptyset$ and, again, $\mathsf{iCCP}(\mathcal{R}_\bot) = \emptyset$, being a 1-CS-CTRS. Also, $\mathsf{CVP}(\mathcal{R}_\bot) = \emptyset$ because all variables in the left-hand sides of the rules in $\mathcal{R}_\bot$ are *frozen* due to $\mu_\bot$. Thus, $\mathsf{ECCP}(\mathcal{R}_\bot) = \emptyset$.

4. Confluence framework

This section describes our confluence framework for proving and disproving (local, strong) confluence of GTRSs. As mentioned in the introduction, the framework is inspired by existing frameworks for proving termination of (variants of) TRSs, starting from [8]. We encapsulate the different stages of confluence proofs for a given GTRS $\mathcal{R}$ as *problems* and the techniques used to treat them and develop the proof as *processors*. Proofs are organized in a *proof tree* whose nodes are the aforementioned problems and whose branches are defined by the (possibly repeated and parallel) use of processors. Section 5 describes a list of processors that can be used in the confluence framework. Forthcoming techniques can often be implemented as a new processor of the framework and then included in an existing strategy for a practical use. In general, it is hard to find a single technique that is able to obtain a complete proof at once. In practice, most proofs are a combination of different (repeatedly used) techniques. This requires the definition of *proof strategies* as a combination of processors. Section 6 describes CONFident's proof strategy.

4.1. Problems

A *problem* is just a structure that contains information used to prove the property we want to analyze. We define some variants of *Confluence* and *Joinability* problems. *Confluence problems* are used to (dis)prove confluence of GTRSs (and also local and strong confluence), and *joinability problems* are used to (dis)prove (strong) joinability of conditional pairs. In the following, we often use τ to refer to a problem when no confusion arises.

Definition 4.1. (Confluence Problems)
Let $\mathcal{R}$ be a GTRS. A (local, strong) *confluence problem*, denoted $CR(\mathcal{R})$ (resp. $WCR(\mathcal{R})$, $SCR(\mathcal{R})$), is *positive* if $\mathcal{R}$ is (locally, strongly) confluent; otherwise, it is *negative*.

Definition 4.2. (Joinability Problems)
Let $\mathcal{R}$ be a GTRS and π be a *conditional pair*. A (strong) *joinability problem*, denoted $JO(\mathcal{R}, \pi)$ (resp. $SJO(\mathcal{R}, \pi)$), is *positive* if π is (strongly) joinable; otherwise, it is *negative*.

In the following, unless established otherwise, our definitions and results pay no attention to the specific type (confluence or joinability) of problems at stake. We just refer to them as "problems".

Remark 4.3. (Relationship between problems)
From well-known results, see, e.g., [3, Section 2.7], the following relations hold for these problems:

If $SCR(\mathcal{R})$ is positive, then $CR(\mathcal{R})$ is positive.
If $CR(\mathcal{R})$ is positive, then $WCR(\mathcal{R})$ is positive.
If $SJO(\mathcal{R}, \pi)$ is positive, then $JO(\mathcal{R}, \pi)$ is positive.

It is well-known that, in general, these implications cannot be reversed.

4.2. Processors

A processor P is a partial function that takes a problem τ as an input and, if P is defined for τ, then it returns either a (possibly empty) set of problems $\tau_1, \ldots, \tau_n$ for some $n \geq 0$ or "no". Usually, $\tau_1, \ldots, \tau_n$ are (hopefully) *simpler* problems. We say that a processor is *sound* if it propagates *positiveness* of *all* returned problems $\tau_1, \ldots, \tau_n$ upwards as *positiveness* of the input problem τ. Symmetrically, a processor is *complete* if *negativeness* of *some* returned problem is propagated upwards as *negativeness* of the input problem τ. Furthermore, if a complete processor returns "no", it tells us that τ is negative and if a sound processor returns an empty set of problems, then τ is trivially positive.

Definition 4.4. A *processor* P is a partial function from problems into sets of problems; alternatively it can return "no". The domain of P (i.e., the set of problems on which P is defined) is denoted $\mathcal{D}om(\mathsf{P})$. We say that P is

- *sound* if for all $\tau \in \mathcal{D}om(\mathsf{P})$, τ is positive whenever $\mathsf{P}(\tau) \neq$ "no" and all $\tau' \in \mathsf{P}(\tau)$ are positive.
- *complete* if for all $\tau \in \mathcal{D}om(\mathsf{P})$, τ is negative whenever $\mathsf{P}(\tau) =$ "no" or some $\tau' \in \mathsf{P}(\tau)$ is negative.

Roughly speaking, soundness is used for *proving* problems *positive*, and completeness is used to prove them *negative*. Sound and complete processors are obviously desirable, as they can be used for both purposes. However, it is often the case that processors which are sound but not complete are available and heavily used (and vice versa) as they implement important techniques for (dis)proving confluence. Section 5 describes several processors and their use in the confluence framework.

4.3. Proofs in the confluence framework

Confluence problems can be proved positive or negative by using a proof tree as follows. Our definitions and results are given, in particular, for confluence problems $CR(\mathcal{R})$ for a GTRS $\mathcal{R}$. They straightforwardly adapt to $WCR(\mathcal{R})$, $SCR(\mathcal{R})$, $JO(\mathcal{R}, \pi)$, and $SJO(\mathcal{R}, \pi)$.

Definition 4.5. (Confluence Proof Tree)
Let $\mathcal{R}$ be a GTRS. A confluence proof tree $\mathcal{T}$ for $\mathcal{R}$ is a tree whose root label is $CR(\mathcal{R})$, whose inner (i.e., non-leaf) nodes are labeled with problems τ, and whose leaves are labeled either with problems τ, or with "yes" or "no". For every inner node n labeled with τ, there is a processor P such that $\tau \in \mathcal{D}om(\mathsf{P})$ and:

1. if $\mathsf{P}(\tau) =$"no" then n has just one child, labeled with "no".
2. if $\mathsf{P}(\tau) = \emptyset$ then n has just one child, labeled with "yes".
3. if $\mathsf{P}(\tau) = \{\tau_1, \ldots, \tau_m\}$ with $m > 0$, then n has m children labeled with the problems $\tau_1, \ldots, \tau_m$.

In this way, a confluence proof tree is obtained by the combination of different processors. The proof of the following result is obvious from the previous definitions.

Theorem 4.6. (Confluence Framework)
Let $\mathcal{R}$ be a GTRS and $\mathcal{T}$ be a confluence proof tree for $\mathcal{R}$. Then:

1. if all leaves in $\mathcal{T}$ are labeled with "yes" and all involved processors are sound for the problems they are applied to, then $\mathcal{R}$ is confluent.
2. if $\mathcal{T}$ has a leaf labeled with "no" and all processors in the path from the root to such a leaf are complete for the problems they are applied to, then $\mathcal{R}$ is not confluent.

Figures 3 and 4 at the end of Setion 5 display examples of proofs obtained by using the confluence framework for some of the examples discussed in this paper.

5. List of processors

In this section, we enumerate some processors for use in the confluence framework, organized according to their functionality. Table 2 displays the complete list, which we develop in the following sections. Table 3 shows which processors (according to their definitions) are able to *finish* a proof branch in a proof tree by either returning an empty set (which is translated by labeling with "yes" a leaf of the tree) or by directly returning "no".

5.1. Cleansing processors

In this section we present a number of processors implementing simple tests to detect and correct particular situations (extra variables, trivial rules, trivial conditions in rules, infeasible conditions, etc.) leading to simplifications of rules or even to an immediate answer. Sometimes, this is done on

Table 2. Available processors

Group	Section	Processors
Cleansing	5.1	$\mathsf{P}_{\mathit{EVar}}$, $\mathsf{P}_{\mathit{Simp}}$, $\mathsf{P}_{\mathit{Inl}}$
Modular decomposition	5.2	P_{MD}
Local/Strong confluence	5.3	P_{HE}
Confluence of CTRSs by transformation	5.4.1–5.4.2	$\mathsf{P}_{\mathcal{U}}$, $\mathsf{P}_{\mathcal{U}_{\text{conf}}}$
Confluence and orthogonality	5.4.3	$\mathsf{P}_{\mathit{Orth}}$
Confluence and local/strong confluence	5.4.4	P_{CR}, P_{WCR}, P_{SCR}
Confluence and (local) confluence of CSR	5.4.5–5.4.6	P_{CanJ}, P_{CnvJ}, P_{CanCR}
Confluence by termination and local confluence	5.4.7	P_{KB}
Joinability	5.5	P_{JO}

Table 3. Ending Processors in the Confluence Framework

Proc.	May end with	Proc.	May end with	Proc.	May end with
$\mathsf{P}_{\mathit{EVar}}$	no	$\mathsf{P}_{\mathit{Orth}}$	yes	P_{KB}	yes
P_{HE}	yes	P_{JO}	yes / no		

the input GTRS, just before attempting a proof; sometimes after applying processors that split the system into components, or that transform the rules to produce new ones exhibiting such problems.

We consider processors $\mathsf{P}_{\mathit{EVar}}$ which checks whether rules with extra variables may definitely imply a non-confluent behavior; $\mathsf{P}_{\mathit{Simp}}$ which removes trivial (components of) rules to simplify them; and $\mathsf{P}_{\mathit{Inl}}$ which tries to obtain substitutions that can be used to remove conditions in rules.

5.1.1. Extra variables check: $\mathsf{P}_{\mathit{EVar}}$

An obvious reason for non-confluence is the presence of extra variables in rules. For instance, a rule $\ell \to x$ for some *extra* variable $x \notin \mathcal{V}ar(\ell)$ 'produces' non-confluence: the following peak is always possible: $x' \leftarrow \ell \to x$ for some fresh variable $x' \notin \mathcal{V}ar(\ell) \cup \{x\}$, but both x and x' are irreducible. In general, as a simple generalization of this fact, given a GTRS $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$, if there is a *feasible* conditional rule $\ell \to r \Leftarrow c \in R$, and a variable $x \in \mathcal{V}ar(r) - \mathcal{V}ar(\ell, c)$ such that $p \in \mathcal{P}os_x(r)$ and for all $q < p$, $root(r|_q)$ is *not a defined symbol*, then $\mathcal{R}$ is not (locally, strongly) confluent. Hence, we define the following processor:

$$\mathsf{P}_{\mathit{EVar}}(CR(\mathcal{R})) = \mathsf{no}$$
$$\mathsf{P}_{\mathit{EVar}}(WCR(\mathcal{R})) = \mathsf{no}$$
$$\mathsf{P}_{\mathit{EVar}}(SCR(\mathcal{R})) = \mathsf{no}$$

if $\mathcal{R}$ contains a rule as above. Then, $\mathsf{P}_{\mathit{EVar}}$ is complete and (trivially) sound.

5.1.2. Simplification: $\mathsf{P}_{\mathit{Simp}}$

The following simplifications of GTRSs are often useful in proofs of confluence problems.

1. *Removing trivial rules.* All rules $t \to t$ or $t \to t \Leftarrow c$ for some term t are *removed.*
2. *Removing trivial conditions.* Conditions $t \approx t$ in the conditional part c of rules $\ell \to r \Leftarrow c$ of J-, O-, or SE-CS-CTRSs are *removed.*
3. *Removing infeasible conditional rules.* Conditional rules $\ell \to r \Leftarrow c$ with an infeasible condition c are *removed*, as they will not be applied in reduction steps.
4. *Removing ground atoms.* Ground atoms B occurring in *feasible* conditions c in clauses $A \Leftarrow c \in H$ or rules $\ell \to r \Leftarrow c \in R$ can be *removed* without affecting the role of the so-simplified clause or rule in computations.

These are applied as much as possible (to each rule in the input system) by means of a *simplifying processor* $\mathsf{P}_{\mathit{Simp}}$:

$$\begin{aligned}\mathsf{P}_{\mathit{Simp}}(\mathit{CR}(\mathcal{R})) &= \{\mathit{CR}(\mathcal{R}')\}\\ \mathsf{P}_{\mathit{Simp}}(\mathit{WCR}(\mathcal{R})) &= \{\mathit{WCR}(\mathcal{R}')\}\\ \mathsf{P}_{\mathit{Simp}}(\mathit{SCR}(\mathcal{R})) &= \{\mathit{SCR}(\mathcal{R}')\}\end{aligned}$$

where $\mathcal{R}' = (\mathcal{F}, \Pi, \mu, H', R')$ is obtained by using the previous transformations to obtain H' and R' from H and R, respectively. Since $\to_{\mathcal{R}}$ and $\to_{\mathcal{R}'}$ coincide, (local, strong) confluence of $\mathcal{R}$ and $\mathcal{R}'$ also coincide. Thus, $\mathsf{P}_{\mathit{Simp}}$ is *sound* and *complete.*

Example 5.1. The following example (#409 in COPS[3]) displays an *oriented* CTRS.[4]

$$\begin{aligned}\mathsf{b} &\to \mathsf{b} && (8)\\ \mathsf{g}(\mathsf{s}(x)) &\to x && (9)\\ \mathsf{h}(\mathsf{s}(x)) &\to x && (10)\\ \mathsf{f}(x,y) &\to \mathsf{g}(\mathsf{s}(x)) \Leftarrow \mathsf{c}(\mathsf{g}(x)) \approx \mathsf{c}(\mathsf{a}) && (11)\\ \mathsf{f}(x,y) &\to \mathsf{h}(\mathsf{s}(x)) \Leftarrow \mathsf{c}(\mathsf{h}(x)) \approx \mathsf{c}(\mathsf{a}) && (12)\end{aligned}$$

Since rule (8) fits the first case above, we can remove it. Thus, we have: $\mathsf{P}_{\mathit{Simp}}(\mathit{CR}(\mathcal{R})) = \{\mathit{CR}(\mathcal{R}')\}$ where $\mathcal{R}'$ consists of the rules $(9), \ldots, (12)$.

5.1.3. Inlining: $\mathsf{P}_{\mathit{Inl}}$

As in [26, Definition 9.4 & Lemma 9.5], the so-called *inlining of rules* is useful to shrink the conditions of *O-rules* in a GTRS $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$. By an *O-rule* we mean a rule $\alpha : \ell \to r \Leftarrow c \in R$ where c consists of conditions which are given the usual *reachability* semantics, by explicitly writing $s \to^* t$, or, in an indirect way, as $s \approx t$ with $\approx$ defined by a single clause $x \approx y \Leftarrow x \to^* y$ in H. For simplicity, in the remainder of the section we assume that the last format is used.

[3]Confluence Problems database, see `https://cops.uibk.ac.at/`

[4]In the following, in order to keep a close connection with the original sources, rather than call them GTRSs, we use TRS, CTRS, CS-CTRS, etc., when citing external examples.

Definition 5.2. (Inlining)
Let $\alpha : \ell \to r \Leftarrow s_1 \approx t_1, \cdots, s_n \approx t_n$ be an O-rule and $t_i = x$ for some variable

$$x \notin \mathcal{V}ar(\ell, s_i, t_1, \ldots, t_{i-1}, t_{i+1}, \ldots t_n) \cup \mathcal{V}ar^{\not\mu}(r, s_1 \ldots, s_n), \tag{13}$$

and $1 \leq i \leq n$. Let $\sigma = \{x \mapsto s_i\}$. The *inlining* of the i-th condition of α with x is

$$\alpha_{x,i} : \ell \to \sigma(r) \Leftarrow \sigma(s_1) \approx t_1, \cdots, \sigma(s_{i-1}) \approx t_{i-1}, \sigma(s_{i+1}) \approx t_{i+1}, \cdots, \sigma(s_n) \approx t_n \tag{14}$$

Given a GTRS $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R \uplus \{\alpha\})$ where α is an O-rule, the *inlining* of the i-th condition of α in $\mathcal{R}$ with x is $\mathcal{R}_{\alpha,x,i} = (\mathcal{F}, \Pi, \mu, H, R \uplus \{\alpha_{x,i}\})$.

For the sake of readability, the proof of the following result is in Appendix A.

Proposition 5.3. Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ be a GTRS, $\alpha \in R$, i, and x as in Definition 5.2. Let s and t be terms.

1. If $s \to_{\mathcal{R}} t$, then $s \to^*_{\mathcal{R}_{\alpha,x,i}} t$.
2. If $s \to_{\mathcal{R}_{\alpha,x,i}} t$, then $s \to_{\mathcal{R}} t$.

Corollary 5.4. Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ be a GTRS, $\alpha \in R$, i, and x as in Definition 5.2. Then, $\to^*_{\mathcal{R}}$ and $\to^*_{\mathcal{R}_{\alpha,x,i}}$ coincide.

Corollary 5.4 entails that confluence of $\mathcal{R}$ and $\mathcal{R}_{\alpha,x,i}$ coincide. By Proposition 5.3, local (resp. strong) confluence of $\mathcal{R}$ implies local (resp. strong) confluence of $\mathcal{R}_{\alpha,x,i}$. In general, though, the opposite direction does *not* hold.

Example 5.5. Consider the following O-CTRS $\mathcal{R}$:

$$\mathsf{b} \to \mathsf{a} \tag{15}$$
$$\mathsf{b} \to x \Leftarrow \mathsf{c} \approx x \tag{16}$$
$$\mathsf{c} \to \mathsf{b} \tag{17}$$
$$\mathsf{c} \to \mathsf{d} \tag{18}$$

Note that $\mathcal{R}$ is *not* locally confluent:

$$\mathsf{a} \;{}_{(15)}\!\leftarrow \mathsf{b} \to_{(16)} \mathsf{d}$$

because the conditional part of (16) is satisfied by $\mathsf{c} \to^*_{\mathcal{R}} \mathsf{d}$. However, the *inlining* of (16) yields the rule $\mathsf{b} \to \mathsf{c}$ which, together with (15), (17), and (18) form the well-known *locally confluent* TRS $\mathcal{R}_{(16),x,1} = \{\mathsf{b} \to \mathsf{a}, \mathsf{b} \to \mathsf{c}, \mathsf{c} \to \mathsf{b}, \mathsf{c} \to \mathsf{d}\}$.

As for P_{Simp}, with P_{Inl} we assume that all rules in the input system $\mathcal{R}$ have been inlined as much as possible to obtain $\mathcal{R}'$. Then we have

$$\begin{array}{rcl} \mathsf{P}_{Inl}(CR(\mathcal{R})) & = & \{CR(\mathcal{R}')\} \\ \mathsf{P}_{Inl}(WCR(\mathcal{R})) & = & \{WCR(\mathcal{R}')\} \\ \mathsf{P}_{Inl}(SCR(\mathcal{R})) & = & \{SCR(\mathcal{R}')\} \end{array}$$

By Corollary 5.4, P_{Inl} is sound and complete for confluence problems $CR(\mathcal{R})$. By Proposition 5.3, P_{Inl} is *complete* (but in general *not sound*, see Example 5.5) for local (resp. strong) confluence problems $WCR(\mathcal{R})$ ($SCR(\mathcal{R})$).

5.2. Modular decomposition: P_{MD}

The *decomposition* of a confluence problem $CR(\mathcal{R})$ into two problems $CR(\mathcal{R}_1)$ and $CR(\mathcal{R}_2)$, by splitting up the input GTRS as $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ for appropriate components or *modules* $\mathcal{R}_1$ and $\mathcal{R}_2$, can be useful in breaking down confluence problems into smaller ones. On this basis, in this section we discuss a processor P_{MD} implementing this approach. Soundness and completeness of P_{MD} can be proved by using existing results about *modularity* of (local, strong) confluence. As modularity of GTRSs has not been investigated yet, in this section we focus on TRSs (as particular GTRSs), see [27] and also [7, Section 8], which we follow here. Our discussion would easily generalize to CTRSs, for which a number of modularity results for confluence are also available, see [28]. In general, a property $\mathcal{P}$ of rewriting-based systems is *modular* if for all systems $\mathcal{R}_1$ and $\mathcal{R}_2$ satisfying $\mathcal{P}$, the union $\mathcal{R}$ of $\mathcal{R}_1$ and $\mathcal{R}_2$ also satisfies $\mathcal{P}$, see [7, Definition 8.1.1]. Usually, properties can be proved modular only if $\mathcal{R}_1$ and $\mathcal{R}_2$ fulfill particular *combination* conditions that are parameterized by Comb: we write $\mathsf{Comb}(\mathcal{R}_1, \mathcal{R}_2)$ to express that $\mathcal{R}_1$ and $\mathcal{R}_2$ satisfy the requirements of a particular combination Comb of modules.

Definition 5.6. (Modularity of confluence)
(Local, Strong) Confluence is called *modular with respect to a given combination* Comb of TRSs (Comb-*modular* for short) if for all TRSs $\mathcal{R}_1 = (\mathcal{F}_1, R_1)$ and $\mathcal{R}_2 = (\mathcal{F}_2, R_2)$ satisfying the condition $\mathsf{Comb}(\mathcal{R}_1, \mathcal{R}_2)$, the following holds: *if* both $\mathcal{R}_1$ and $\mathcal{R}_2$ are (locally, strongly) confluent, then the union $\mathcal{R}_1 \cup \mathcal{R}_2 = (\mathcal{F}_1 \cup \mathcal{F}_2, R_1 \cup R_2)$ is also (locally, strongly) confluent.

For TRSs $\mathcal{R}$, processor P_{MD} tries to find such a decomposition:

$$\begin{array}{rcl} \mathsf{P}_{MD}(CR(\mathcal{R})) & = & \{CR(\mathcal{R}_1), CR(\mathcal{R}_2)\} \\ \mathsf{P}_{MD}(WCR(\mathcal{R})) & = & \{WCR(\mathcal{R}_1), WCR(\mathcal{R}_2)\} \\ \mathsf{P}_{MD}(SCR(\mathcal{R})) & = & \{SCR(\mathcal{R}_1), SCR(\mathcal{R}_2)\} \end{array}$$

if $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, $\mathsf{Comb}(\mathcal{R}_1, \mathcal{R}_2)$ holds and (local, strong) confluence is Comb-modular. This definition guarantees *soundness* of P_{MD} on (local, strong) confluence problems as it is implied by the modularity of the corresponding property. Thus, P_{MD} is *sound* on (local, strong) confluence problems if $\mathsf{Comb}(\mathcal{R}_1, \mathcal{R}_2)$ holds and (local, strong) confluence is Comb-modular.

5.2.1. Modular combinations and modularity results for TRSs

In the literature, a number of *combinations* $\mathsf{Comb}(\mathcal{R}_1,\mathcal{R}_2)$ of TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ have been considered to prove modularity. In particular,

- *disjoint* combinations (where $\mathcal{R}_1$ and $\mathcal{R}_2$ share no function symbol [29]),
- *constructor-sharing* combinations (where $\mathcal{R}_1$ and $\mathcal{R}_2$ may share constructor symbols only [30]),
- *composable* combinations (where $\mathcal{R}_1$ and $\mathcal{R}_2$ may share constructor symbols and also defined symbols provided that they also share the rules defining them [31]).

See also [7, Definition 8.1.4] for definitions of all these combinations of TRSs, which we refer as $\mathsf{DisjU}(\mathcal{R}_1,\mathcal{R}_2)$, $\mathsf{CShC}(\mathcal{R}_1,\mathcal{R}_2)$, and $\mathsf{CompC}(\mathcal{R}_1,\mathcal{R}_2)$, respectively. Note that

$$\mathsf{DisjU}(\mathcal{R}_1,\mathcal{R}_2) \Rightarrow \mathsf{CShC}(\mathcal{R}_1,\mathcal{R}_2) \Rightarrow \mathsf{CompC}(\mathcal{R}_1,\mathcal{R}_2) \tag{19}$$

see [7, Figure 8.2]. In Table 4 we show an excerpt of the results displayed in [7, Table 8.1] regarding modularity of confluence, local confluence, and strong confluence of TRSs. In this table, we use the notion of *layer-preserving TRS* (LP) [32, Definition 5.5]: in a composable combination, i.e., $\mathsf{CompC}(\mathcal{R}_1,\mathcal{R}_2)$ holds, let $\mathcal{B} = \mathcal{F}_1 \cap \mathcal{F}_2$ be the set of *shared function symbols* and, for $i \in \{1,2\}$, $\mathcal{A}_i = \mathcal{F}_i - \mathcal{B}$ be the *alien* symbols for $\mathcal{R}_{3-i}$. Let $i \in \{1,2\}$. Then, $\mathcal{R}_i = (\mathcal{F}_i, R_i)$ is called *layer-preserving* if for all $\ell \to r \in \mathcal{R}_i$, we have $root(r) \in \mathcal{A}_i$ whenever $root(\ell) \in \mathcal{A}_i$. A *constructor-sharing* union is *layer-preserving* if $\mathcal{R}_1$ and $\mathcal{R}_2$ contain neither collapsing rules nor constructor-lifting rules (i.e., rules $\ell \to r$ such that $root(r)$ is a shared constructor).

Table 4. Excerpt of [7, Table 8.1]. Here L is *linearity*, LL is *left-linearity*, and LP is *layer preservation*

Property	Disjoint union	Constructor-sharing	Composable
Confluence	[29, Coro. 4.1]	+LP [33, Coro. 5.11]	+LP [34]
		+LL [35], see [7, Coro. 8.6.38(1)]	
Local Conf.	[36]	[36]	[36]
Strong Conf.	+L [37]	+L [37]	+L [37]

Remark 5.7. (Comments on Table 4)
As remarked in [7, Sections 8.2.1 & 8.6.3], for composable combinations (hence for disjoint unions and constructor-sharing combinations, see (19)), the following *non interfering* property [36] holds:

$$\mathsf{CP}(\mathcal{R}_1 \cup \mathcal{R}_2) \quad = \quad \mathsf{CP}(\mathcal{R}_1) \cup \mathsf{CP}(\mathcal{R}_2) \tag{20}$$

that is, the set of critical pairs of the union is the union of the critical pairs of the components.

Accordingly, [7] makes the following observations that justify the last two rows in Table 4:

- Middeldorp proved that *local confluence is modular* for disjoint unions of TRSs, see, e.g., [28, Theorem 2.4], originally in [36]. Ohlebusch observes that local confluence is modular for any combination of TRSs satisfying (20) [7, page 249, penultimate paragraph]. Thus, local confluence is modular for constructor-sharing and composable combinations too, see also [7, Corollary 8.6.41(1)].

- As explained in the paragraph below [7, Example 8.2.2], the results about *modularity of strong confluence*, not explicit in [37], are a consequence of [37, Lemma 3.2] (*A linear TRS is strongly closed iff it is strongly confluent*) and the *non interfering* property (20) for composable (hence disjoint, sharing constructor) combinations.

5.2.2. Soundness of P_{MD}

As a consequence of the discussion in Section 5.2.1, (see Table 4 and Remark 5.7), given $\mathcal{R}$, $\mathcal{R}_1$, and $\mathcal{R}_2$ such that $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, P_{MD} is *sound* for

- $CR(\mathcal{R})$ if (i) $\mathsf{DisjU}(\mathcal{R}_1, \mathcal{R}_2)$ holds, or (ii) $\mathsf{CShC}(\mathcal{R}_1, \mathcal{R}_2)$ holds and $\mathcal{R}$ is *left-linear*, or (iii) $\mathsf{CShC}(\mathcal{R}_1, \mathcal{R}_2)$ or $\mathsf{CompC}(\mathcal{R}_1, \mathcal{R}_2)$ holds and $\mathcal{R}_1$ and $\mathcal{R}_2$ are *layer preserving*.

- $WCR(\mathcal{R})$ if $\mathsf{CompC}(\mathcal{R}_1, \mathcal{R}_2)$ (and hence $\mathsf{DisjU}(\mathcal{R}_1, \mathcal{R}_2)$ or $\mathsf{CShC}(\mathcal{R}_1, \mathcal{R}_2)$) holds.

- $SCR(\mathcal{R})$ if $\mathcal{R}$ is linear, and $\mathsf{DisjU}(\mathcal{R}_1, \mathcal{R}_2)$ or $\mathsf{CShC}(\mathcal{R}_1, \mathcal{R}_2)$ or $\mathsf{CompC}(\mathcal{R}_1, \mathcal{R}_2)$ holds.

Example 5.8. Consider the following TRS [38, Example 4]:

$$
\begin{aligned}
\mathsf{nats} &\to \mathsf{from}(\mathsf{0}) && (21)\\
\mathsf{inc}(x{:}y) &\to \mathsf{s}(x){:}\mathsf{inc}(y) && (22)\\
\mathsf{hd}(x{:}y) &\to x && (23)\\
\mathsf{tl}(x{:}y) &\to y && (24)\\
\mathsf{from}(x) &\to x{:}\mathsf{from}(\mathsf{s}(x)) && (25)\\
\mathsf{inc}(\mathsf{tl}(\mathsf{from}(x))) &\to \mathsf{tl}(\mathsf{inc}(\mathsf{from}(x))) && (26)
\end{aligned}
$$

With $\mathcal{R}_1 = \{(23)\}$ and $\mathcal{R}_2 = \{(21), (22), (24), (25), (26)\}$, with $\_{:}\_$ the only shared constructor symbol, $\mathsf{CShC}(\mathcal{R}_1, \mathcal{R}_2)$ holds. Since, $\mathcal{R}$ is *left-linear*, $\mathsf{P}_{MD}(CR(\mathcal{R})) = \{CR(\mathcal{R}_1), CR(\mathcal{R}_2)\}$.

5.2.3. Completeness of P_{MD}

Regarding modularity of disjoint unions $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, where $\mathsf{DisjU}(\mathcal{R}_1, \mathcal{R}_2)$ holds, [29, Corollary 4.1] proves that *$\mathcal{R}$ is confluent iff both $\mathcal{R}_1$ and $\mathcal{R}_2$ are*. This entails that P_{MD} is *complete* on confluence problems for disjoint unions.

Also, P_{MD} is *complete* on local confluence problems for disjoint unions: [28, Theorem 2.4] establishes modularity of local confluence and, according to [28, Definition 2.2], local confluence is modular for a disjoint union of TRSs if the following equivalence holds: $\mathcal{R}_1 \cup \mathcal{R}_2$ is locally confluent iff both $\mathcal{R}_1$ and $\mathcal{R}_2$ are locally confluent. The point is that, for disjoint unions, joinability of critical pairs in $\mathsf{CP}(\mathcal{R}_i)$ for $i \in \{1,2\}$ cannot depend on reductions using $\mathcal{R}_{3-i}$. Since (20) holds for disjoint unions, it follows that local confluence of $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ (i.e., joinability of all critical pairs in $\mathsf{CP}(\mathcal{R}_1 \cup \mathcal{R}_2)$) implies local confluence of both $\mathcal{R}_1$ and $\mathcal{R}_2$. For composable combinations (hence for constructor sharing combinations), Ohlebusch provides a similar treatment in [32]: [32, Proposition 5.3(1)] establishes modularity of local confluence for composable combinations, and, for composable combinations, $\mathcal{R}_1 \cup \mathcal{R}_2$ is locally confluent iff both $\mathcal{R}_1$ and $\mathcal{R}_2$ are locally confluent [32, Definition 3.2]. Similarly, since strong confluence is characterized by the strong joinability of critical pairs, P_{MD} is also *complete* on strong confluence problems for disjoint unions.

5.3. Processor for local/strong confluence problems: P_{HE}

Processor P_{HE} treats local and strong confluence problems as (strong) joinability of conditional critical and variable pairs.

Local confluence. Extended conditional critical pairs $\mathsf{ECCP}(\mathcal{R})$ enable the following characterizations of local confluence of GTRSs (and, in particular, of CS-TRSs, CTRSs, CS-CTRSs, etc.), thus extending the well-known result for TRSs by Huet [37, Lemma 3.1].

Theorem 5.9. (Local confluence of GTRSs)
Let $\mathcal{R}$ be a GTRS. Then,

1. $\mathcal{R}$ is locally confluent iff each $\pi \in \mathsf{ECCP}(\mathcal{R})$ is joinable [1, Theorem 62].
2. If $\mathcal{R}$ is an SE-CS-CTRS such that (†) for all $\langle s,t\rangle \Leftarrow x \to x', c \in \mathsf{CVP}(\mathcal{R})$, and $u \approx v \in c$, $x \notin \mathcal{V}ar^{\not\mu}(u) \cup \mathcal{V}ar^{\not\mu}(v)$, then, $\mathcal{R}$ is locally confluent iff each $\pi \in \mathsf{pCCP}(\mathcal{R}) \cup \mathsf{iCCP}(\mathcal{R})$ is joinable [1, Corollary 63].
3. If $\mathcal{R}$ is a CS-TRS, then $\mathcal{R}$ is locally confluent iff each $\pi \in \mathsf{CP}(\mathcal{R},\mu) \cup \mathsf{LHCP}(\mathcal{R},\mu)$ is joinable [25, Theorem 30].

Accordingly, for GTRSs $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$,

$$\mathsf{P}_{HE}(\mathit{WCR}(\mathcal{R})) = \{\mathit{JO}(\mathcal{R},\pi_1),\ldots,\mathit{JO}(\mathcal{R},\pi_n)\}, \text{where}$$
$$\{\pi_1,\ldots,\pi_n\} = \begin{cases} \mathsf{CP}(\mathcal{R}) & \text{if } \mathcal{R} \text{ is a TRS} \\ \mathsf{CP}(\mathcal{R},\mu) \cup \mathsf{LHCP}(\mathcal{R},\mu) & \text{if } \mathcal{R} \text{ is a CS-TRS} \\ \mathsf{pCCP}(\mathcal{R}) \cup \mathsf{iCCP}(\mathcal{R}) & \text{if } \mathcal{R} \text{ is an SE-CS-CTRS satisfying (†)} \\ \mathsf{ECCP}(\mathcal{R}) & \text{otherwise} \end{cases}$$

Example 5.10. Consider the CTRS $\mathcal{R}$ in Example 1.1 with $\mathsf{ECCP}(\mathcal{R}) = \{(6),(7)\}$ (see Example 3.5). Thus, we have $\mathsf{P}_{HE}(\mathit{WCR}(\mathcal{R})) = \{\mathit{JO}(\mathcal{R},(6)), \mathit{JO}(\mathcal{R},(7))\}$.

Example 5.11. (Continuing Examples 1.1 and 3.6)
For the CS-CTRS $\mathcal{R}_\perp$ in Example 1.1, $\mathsf{ECCP}(\mathcal{R}_\perp) = \emptyset$. Thus, $\mathsf{P}_{HE}(\mathit{WCR}(\mathcal{R}_\perp)) = \emptyset$.

Note that, by Theorem 5.9, P_{HE} is *sound and complete* on local confluence problems.

Strong confluence. Linear TRSs whose critical pairs are *strongly joinable* are strongly confluent, [3, Lemma 6.3.3]. Accordingly, for *linear* TRSs $\mathcal{R}$,

$$\mathsf{P}_{HE}(SCR(\mathcal{R})) = \{SJO(\mathcal{R},\pi_1),\ldots,SJO(\mathcal{R},\pi_n)\}, \text{where } \{\pi_1,\ldots,\pi_n\} = \mathsf{CP}(\mathcal{R})$$

This processor is *sound* and *complete* on strong confluence problems for linear TRSs: the existence of a non-strongly-joinable conditional pair witnesses non-strong-joinabilty.

Table 5 summarizes the use of P_{EVar}, P_{Simp}, P_{Inl}, P_{MD}, and P_{HE} in the confluence framework.

Table 5. Use of P_{EVar}, P_{Simp}, P_{Inl}, P_{MD} and P_{HE} in the confluence framework

Problem	P_{EVar}	P_{Simp}	P_{Inl}	P_{MD}	P_{HE}
CR	✓	✓	✓	✓	
WCR	✓	✓	✓	✓	✓
SCR	✓	✓	✓	✓	✓
TRSs	✓	✓		✓	✓
CS-TRSs	✓	✓			✓
CTRSs	✓	✓	✓		✓
CS-CTRSs	✓	✓	✓		✓
GTRSs	✓	✓	✓		✓

P_{EVar}, P_{Simp}, and P_{HE} are sound and complete. P_{Inl} is complete, but it is sound on CR problems only. P_{MD} is sound; if P_{MD} is used with *disjoint unions* of TRSs, then it is *complete* on all problems it applies to; if P_{MD} is used with *composable combinations* of TRSs (hence, disjoint unions and constructor-sharing combinations), then it is *complete* on WCR problems.

Table 6. Use of Processors for Confluence Problems in the confluence framework

Problem	$\mathsf{P}_{\mathcal{U}}$	$\mathsf{P}_{\mathcal{U}_{\text{conf}}}$	P_{Orth}	P_{CR}	P_{WCR}	P_{SCR}	P_{CanJ}	P_{CnvJ}	P_{CanCR}	P_{KB}
CR	✓	✓	✓		✓	✓	✓	✓	✓	✓
WCR	✓	✓	✓	✓		✓	✓	✓	✓	
SCR				✓	✓					
TRSs			✓	✓	✓	✓	✓	✓	✓	✓
CS-TRSs			✓	✓	✓	✓				✓
CTRSs	✓	✓	✓	✓	✓	✓				✓
CS-CTRSs				✓	✓	✓				✓
GTRSs				✓	✓	✓				✓
Sound	✓	✓	✓			✓	✓	✓	✓	✓
Complete			✓		✓					✓

P_{CR} is *sound* on WCR problems, and *complete* on SCR-problems.

5.4. Processors for confluence problems

In this section we discuss a number of processors to be used with confluence problems. Processors $\mathsf{P}_{\mathcal{U}}$ and $\mathsf{P}_{\mathcal{U}_{\text{conf}}}$ treat confluence problems for CTRSs $\mathcal{R}$ by transforming them into TRSs $\mathcal{U}(\mathcal{R})$ and $\mathcal{U}_{\text{conf}}(\mathcal{R})$ and then solving the corresponding confluence problem. Following the results obtained by Huet for left-linear TRSs without critical pairs [37], processor P_{Orth} exploits (variants of) *orthogonality* investigated for CS-TRSs and CTRSs to treat confluence problems. Processors P_{CR}, P_{WCR} and P_{SCR} relate confluence problems and local/strong confluence problems. Processors P_{CanJ} and P_{CnvJ} translate confluence problems for TRSs into local confluence problems of CS-TRSs. Processor P_{CanCR} translates confluence problems for TRSs into confluence problems for CS-TRSs. Finally, for terminating GTRSs $\mathcal{R}$, P_{KB} either translates confluence problems into joinability problems of an appropriate set of conditional critical pairs, or else into a local confluence problem for $\mathcal{R}$. Table 6 summarizes the use of processors for confluence problems in the confluence framework.

5.4.1. Confluence of terminating CTRSs as confluence of TRSs: $\mathsf{P}_{\mathcal{U}}$

An oriented CTRS $\mathcal{R}$ is *deterministic* (DCTRS) if for every rule $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$ in $\mathcal{R}$ and every $1 \leq i \leq n$, we have $\mathcal{V}ar(s_i) \subseteq \mathcal{V}ar(\ell) \cup \bigcup_{j=1}^{i-1} \mathcal{V}ar(t_j)$ [7, Def. 7.2.33]. In the following, $\mathcal{U}$ is the transformation in [7, Def. 7.2.48] that, given a 3-DCTRSs $\mathcal{R} = (\mathcal{F}, R)$ obtains a TRS $\mathcal{U}(\mathcal{R})$. Each rule $\alpha : \ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n \in R$ is transformed into $n+1$ unconditional rules:

$$\begin{array}{rcll} \ell & \to & U_1^\alpha(s_1, \vec{x}_1) & \\ U_{i-1}^\alpha(t_{i-1}, \vec{x}_{i-1}) & \to & U_i^\alpha(s_i, \vec{x}_i) & \quad 2 \leq i \leq n \\ U_n^\alpha(t_n, \vec{x}_n) & \to & r & \end{array}$$

where U_i^α are fresh new symbols and $\vec{x}_i$ are sequences of variables in $\mathcal{V}ar(\ell) \cup \mathcal{V}ar(t_1) \cup \cdots \cup \mathcal{V}ar(t_{i-1})$ for $1 \leq i \leq n$. Unconditional rules remain unchanged. Then, $\mathcal{U}(\mathcal{R}) = (\mathcal{U}(\mathcal{F}), \mathcal{U}(R))$, where $\mathcal{U}(\mathcal{F})$ is the signature $\mathcal{F}$ extended with the new symbols introduced by transformation $\mathcal{U}$ and $\mathcal{U}(R)$ is the new set of rules obtained from R as explained above.

Example 5.12. (Transformation $\mathcal{U}$)
For the DCTRS $\mathcal{R} = (\mathcal{F}, R)$ with $R = \{\mathsf{a} \to \mathsf{b} \Leftarrow \mathsf{a} \approx \mathsf{b}, \mathsf{a} \to \mathsf{b} \Leftarrow \mathsf{a} \approx \mathsf{c}\}$, $\mathcal{U}(R)$ consists of:

$$\begin{array}{rcll} \mathsf{a} & \to & U(\mathsf{a}) & (27) \\ U(\mathsf{b}) & \to & \mathsf{b} & (28) \\ \mathsf{a} & \to & U'(\mathsf{a}) & (29) \\ U'(\mathsf{c}) & \to & \mathsf{b} & (30) \end{array}$$

Note that $\mathcal{U}(\mathcal{F}) = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, U, U'\}$.

Remark 5.13. (Simulating conditional rewriting by unconditional rewriting)
Starting from the pioneering work by Marchiori [39], several authors have investigated the ability of transformations from CTRSs $\mathcal{R}$ to TRSs to *simulate* conditional rewriting by means of unconditional rewriting, see, e.g., [40, 41] and the references therein. It is well-known that $\mathcal{U}$ is *simulation-complete*,

i.e., $\to_{\mathcal{R}} \subseteq \to^*_{\mathcal{U}(\mathcal{R})}$ see [41, Section 3]; however, $\mathcal{U}$ is *not simulation-sound*: there are terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $s \to^*_{\mathcal{U}(\mathcal{R})} t$ but $s \not\to^*_{\mathcal{R}} t$, see, e.g., [41, Example 3.3].

The following definition prepares Theorem 5.16 below, which enables the use of transformation $\mathcal{U}$ in the confluence framework.

Definition 5.14. Let $\mathcal{R}$ be a 3-DCTRS. We say that $\mathcal{U}$ preserves $\to_{\mathcal{R}}$-irreducibility of a 3-DCTRS $\mathcal{R} = (\mathcal{F}, R)$ if for all terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, if t is $\to_{\mathcal{R}}$-irreducible, then t is $\to_{\mathcal{U}(\mathcal{R})}$-irreducible.

In general, this property does *not* hold for all 3-DCTRSs.

Example 5.15. Let $\mathcal{R}$ and $\mathcal{U}(\mathcal{R})$ as in Example 5.12. Although term a is $\to_{\mathcal{R}}$-irreducible, it is *not* $\to_{\mathcal{U}(\mathcal{R})}$-irreducible as, e.g., $\mathsf{a} \to_{\mathcal{U}(\mathcal{R})} U(\mathsf{a})$. Thus, $\mathcal{U}$ does *not* preserve $\to_{\mathcal{R}}$-irreducibility of $\mathcal{R}$.

Theorem 5.16. Let $\mathcal{R}$ be a 3-DCTRS. If $\mathcal{R}$ is terminating, $\mathcal{U}$ preserves $\to_{\mathcal{R}}$-irreducibility, and $\mathcal{U}(\mathcal{R})$ is confluent, then $\mathcal{R}$ is confluent.[5]

Proof:
By contradiction. If $\mathcal{R}$ is not confluent, then there are terms $s, t, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $s \to^*_{\mathcal{R}} t$ and $s \to^*_{\mathcal{R}} t'$ but t and t' are not $\to^*_{\mathcal{R}}$-joinable. By termination of $\to_{\mathcal{R}}$, we can assume that t and t' are different $\to_{\mathcal{R}}$-irreducible terms, i.e., $t \neq t'$. By confluence of $\mathcal{U}(\mathcal{R})$, $t \to^*_{\mathcal{U}(\mathcal{R})} u$ and $t' \to^*_{\mathcal{U}(\mathcal{R})} u$ for some term u. Since $\mathcal{U}$ preserves $\to_{\mathcal{R}}$-irreducibility, both t and t' are $\to_{\mathcal{U}(\mathcal{R})}$-irreducible. Thus, $t = u = t'$, a contradiction. □

For a practical use of Theorem 5.16, we introduce a sufficient condition for preservation of irreducibility. In the following, given a sequence of expressions c and a set of variables V, $c^{\downarrow_V}$ is the *partial grounding* of c where all variables $x \in \mathcal{V}ar(c) \cap V$ are replaced by c_x. The following result provides a *sufficient condition* for preservation of $\to_{\mathcal{R}}$-irreducibility. Intuitively, requiring *feasibility* of $c^{\downarrow_{\mathcal{V}ar(\ell)}}$ for all rules $\ell \to r \Leftarrow c$ guarantees that no particular instantiation of variables due to pattern matching against ℓ plays a role in the satisfaction of the conditional part c of the rule.

Proposition 5.17. Let $\mathcal{R} = (\mathcal{F}, R)$ be a 3-DCTRS. If for all $\ell \to r \Leftarrow c \in R$, $c^{\downarrow_{\mathcal{V}ar(\ell)}}$ is $\overline{\mathcal{R}}$-feasible, then $\mathcal{U}$ preserves $\to_{\mathcal{R}}$-irreducibility

Proof:
If $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is $\to_{\mathcal{R}}$-irreducible, we consider two cases: First, if there is no $p \in \mathcal{P}os(t)$ such that $t|_p = \sigma(\ell)$ for some $\ell \to r \Leftarrow c \in R$ and substitution σ, then t is also $\to_{\mathcal{U}(\mathcal{R})}$-irreducible, as the left-hand sides of rules in $\mathcal{U}(\mathcal{R})$ either coincide with those in $\mathcal{R}$ or include a symbol in $\mathcal{U}(\mathcal{F}) - \mathcal{F}$. Otherwise, since t is $\to_{\mathcal{R}}$-irreducible, $\sigma'(c)$ must be $\overline{\mathcal{R}}$-infeasible for all substitutions σ' such that $\sigma'(x) = \sigma(x)$ for all $x \in \mathcal{V}ar(\ell)$. However, $c^{\downarrow_{\mathcal{V}ar(\ell)}}$ is $\overline{\mathcal{R}}$-feasible, i.e., there is a substitution ς such that $\varsigma(c^{\downarrow_{\mathcal{V}ar(\ell)}})$ holds. Thus the substitution σ' defined by $\sigma'(x) = \sigma(x)$ for all $x \in \mathcal{V}ar(\ell)$ and $\sigma'(y) = \varsigma(y)$ for all $y \notin \mathcal{V}ar(\ell)$ also satisfies $\sigma(c)$, a contradiction. □

[5]This result fixes the buggy [5, Theorem 8].

Example 5.18. Consider the following 3-DCTRS $\mathcal{R}$:

$$\begin{array}{rcll} \mathsf{g}(x) & \to & x & (31) \\ \mathsf{f}(x) & \to & x \Leftarrow \mathsf{g}(x) \approx x & (32) \end{array}$$

Since $\mathsf{g}(c_x) \approx c_x$ is clearly *feasible* (we have $\underline{\mathsf{g}(c_x)} \to_{(31)} c_x$), $\mathcal{U}$ preserves $\to_{\mathcal{R}}$-irreducibility.

Processor $\mathsf{P}_{\mathcal{U}}$ transforms a confluence problem for a terminating 3-DCTRS $\mathcal{R}$ into a confluence problem for a TRS $\mathcal{U}(\mathcal{R})$. If $\mathcal{R}$ is a terminating 3-DCTRS such that $\mathcal{U}$ preserves $\to_{\mathcal{R}}$-irreducibility, then

$$\mathsf{P}_{\mathcal{U}}(CR(\mathcal{R})) \;=\; \{CR(\mathcal{U}(\mathcal{R}))\}$$

By relying on Theorem 5.16, $\mathsf{P}_{\mathcal{U}}$ is *sound*. However, it is *not* complete.

Example 5.19. ($\mathsf{P}_{\mathcal{U}}$ is not complete)
The 3-DCTRS $\mathcal{R}$ in Example 5.12 is (locally) confluent: the rules are infeasible; thus, the one-step reduction relation is empty. However, $\mathcal{U}(\mathcal{R})$ defines a non-joinable peak $U(\mathsf{a})\; {}_{\mathcal{U}(\mathcal{R})}\!\leftarrow \mathsf{a} \to_{\mathcal{U}(\mathcal{R})} U'(\mathsf{a})$.

5.4.2. Confluence of CTRSs as confluence of TRSs using an improved transformation: $\mathsf{P}_{\mathcal{U}_{\text{conf}}}$

Gmeiner, Nishida and Gramlich [42] introduced transformation $\mathcal{U}_{\text{conf}}$, which can also be used in proofs of confluence of 3-*DCTRSs* in the confluence framework. Each rule $\alpha : \ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$ is transformed into $n+1$ unconditional rules [42, Definition 6]:

$$\begin{array}{rcl} \ell & \to & U_{\ell,s_1}(s_1, \vec{x}_1) \\ U_{\ell,s_1}(t_1, \vec{x}_1) & \to & U_{\ell,s_1,t_1,s_2}(s_2, \vec{x}_2) \\ \vdots & & \\ U_{\ell,s_1,t_1,\ldots,s_n}(t_n, \vec{x}_n) & \to & r \end{array}$$

where new symbols $U_{\ell,\ldots}$ depending on the the left-hand side ℓ of α and also on the terms occurring in the conditions of α are introduced. For each $1 \leq i \leq n$, $\vec{x}_i$ is as in transformation $\mathcal{U}$. Then, $\mathcal{U}_{\text{conf}}(\mathcal{R}) = (\mathcal{U}_{\text{conf}}(\mathcal{F}), \mathcal{U}_{\text{conf}}(R))$, where $\mathcal{U}_{\text{conf}}(\mathcal{F})$ is the signature $\mathcal{F}$ extended with the new symbols introduced by transformation $\mathcal{U}_{\text{conf}}$ and $\mathcal{U}_{\text{conf}}(R)$ is the new set of rules obtained from R.

Example 5.20. (Transformation $\mathcal{U}_{\text{conf}}$)
For $\mathcal{R} = (\mathcal{F}, R)$ in Example 5.12, $\mathcal{U}_{\text{conf}}(R)$ consists of the rules:

$$\begin{array}{rcll} \mathsf{a} & \to & U(\mathsf{a}) & (33) \\ U(\mathsf{b}) & \to & \mathsf{b} & (34) \\ U(\mathsf{c}) & \to & \mathsf{b} & (35) \end{array}$$

Compared with $\mathcal{U}(\mathcal{R})$ in Example 5.12, note that U' is missing thanks to the refined definition of $\mathcal{U}_{\text{conf}}$.

A DCTRS is *weakly left-linear* if "variables that occur more than once in the lhs of a conditional rule and the rhs's of conditions should not occur at all in lhs's of conditions or the rhs of the conditional rule" [40, Definition 3.17]. Processor $\mathsf{P}_{\mathcal{U}_{\mathrm{conf}}}$ transforms a confluence problem for a 3-DCTRS $\mathcal{R}$ into a confluence problem for a TRS $\mathcal{U}_{\mathrm{conf}}(\mathcal{R})$, where $\mathcal{U}_{\mathrm{conf}}$ is the transformation in [42, Definition 6]:

$$\mathsf{P}_{\mathcal{U}_{\mathrm{conf}}}(CR(\mathcal{R})) = \{CR(\mathcal{U}_{\mathrm{conf}}(\mathcal{R}))\}$$

if $\mathcal{R}$ is a weakly left-linear 3-DCTRS. By [42, Theorem 9], $\mathsf{P}_{\mathcal{U}_{\mathrm{conf}}}$ is sound; however, it is *not* complete.

Example 5.21. **($\mathsf{P}_{\mathcal{U}_{\mathrm{conf}}}$ is not complete)**
For the 3-DCTRS $\mathcal{R} = (\mathcal{F}, R)$ with $R = \{\mathsf{a} \to \mathsf{b} \Leftarrow \mathsf{b} \approx \mathsf{a}, \mathsf{a} \to \mathsf{b} \Leftarrow \mathsf{c} \approx \mathsf{a}\}$, we have

$$\mathcal{U}(\mathcal{R}) = \mathcal{U}_{\mathrm{conf}}(\mathcal{R}) = \{\mathsf{a} \to U(\mathsf{b}), U(\mathsf{a}) \to \mathsf{b}, \mathsf{a} \to U'(\mathsf{c}), U'(\mathsf{a}) \to \mathsf{b}\}.$$

The rules of $\mathcal{R}$ are infeasible and can be removed. Thus, $\to_{\mathcal{R}}$ is *empty*, hence confluent. However, the peak $U(\mathsf{b}) \;{}_{\mathcal{U}_{\mathrm{conf}}(\mathcal{R})}\!\leftarrow \mathsf{a} \to_{\mathcal{U}_{\mathrm{conf}}(\mathcal{R})} U'(\mathsf{c})$ is not joinable, as both $U(\mathsf{b})$ and $U'(\mathsf{c})$ are $\mathcal{U}_{\mathrm{conf}}(\mathcal{R})$-irreducible.

Note that $\mathcal{R}$ in Example 5.12 can be proved confluent using $\mathcal{U}_{\mathrm{conf}}(\mathcal{R})$: as shown in Example 5.20, $\mathcal{U}_{\mathrm{conf}}(\mathcal{R})$ is orthogonal, hence confluent, which proves confluence of $\mathcal{R}$ by [42, Theorem 9]. However, the following example shows that $\mathsf{P}_{\mathcal{U}}$ can be used to prove confluence when $\mathsf{P}_{\mathcal{U}_{\mathrm{conf}}}$ fails.

Example 5.22. The 3-DCTRS $\mathcal{R}$ in Example 5.18 is clearly terminating and, as shown in the example, $\mathcal{U}$ preserves $\to_{\mathcal{R}}$-irreducibility. The TRS $\mathcal{U}(\mathcal{R})$:

$$\mathsf{g}(x) \to x \tag{36}$$
$$\mathsf{f}(x) \to U(\mathsf{g}(x), x) \tag{37}$$
$$U(x, x) \to x \tag{38}$$

is terminating and has no critical pair. Hence, $\mathcal{U}(\mathcal{R})$ is confluent. By Theorem 5.16, $\mathcal{R}$ is confluent. Note that $\mathcal{R}$ is *not* weakly left-linear. Thus, $\mathcal{U}_{\mathrm{conf}}(\mathcal{R})$ cannot be used to prove confluence of $\mathcal{R}$.

Thus, $\mathsf{P}_{\mathcal{U}}$ and $\mathsf{P}_{\mathcal{U}_{\mathrm{conf}}}$ are complementary. However, our experiments show that $\mathsf{P}_{\mathcal{U}_{\mathrm{conf}}}$ applies more frequently than $\mathsf{P}_{\mathcal{U}}$, see Table 11.

5.4.3. Orthogonality: P_{Orth}

A GTRS $\mathcal{R}$ is *left-linear* if for all rules $\ell \to r \Leftarrow c$, the left-hand side ℓ is linear.

- A left-linear TRS $\mathcal{R}$ whose critical pairs are all trivial is called *weakly orthogonal*. For left-linear TRSs, Huet provided several results [37, Section 3.3] leading, in particular, to conclude that *weakly orthogonal TRSs are confluent*, see [3, Section 6.4].

- A left-linear CS-TRS $(\mathcal{R}, \mu)$ is μ-orthogonal if $\mathsf{CP}(\mathcal{R}, \mu) = \mathsf{LHCP}(\mathcal{R}, \mu) = \emptyset$ [25, Definition 35]. By [25, Corollary 36], μ-orthogonal CS-TRSs are confluent.

- A left-linear CTRS is (almost) *orthogonal* if $\mathsf{pCCP}(\mathcal{R}) = \emptyset$ (resp. $\mathsf{pCCP}(\mathcal{R})$ consists of *trivial* pairs $\langle t, t\rangle \Leftarrow c$ with critical position $p = \Lambda$) [7, Definition 7.1.10(1 & 2)]. By [7, Theorem 7.4.14],[6] orthogonal, properly oriented, and right-stable 3-CTRS are confluent, where
  - A 3-CTRS $\mathcal{R}$ is *properly oriented* if every rule $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$ satisfies: if $\mathcal{V}ar(r) \notin \mathcal{V}ar(\ell)$, then $\mathcal{V}ar(s_i) \subseteq \mathcal{V}ar(\ell) \cup \bigcup_{j=1}^{i-1} \mathcal{V}ar(t_j)$ for all $1 \leq i \leq n$. [7, Definition 7.4.13].
  - A CTRS is *right-stable* if for every rule $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n \in \mathcal{R}$ and for all $1 \leq i \leq n$, (a) $(\mathcal{V}ar(\ell) \cup \bigcup_{j=1}^{i-1} \mathcal{V}ar(s_j \approx t_j) \cup \mathcal{V}ar(s_i)) \cap \mathcal{V}ar(t_i) = \emptyset$, and (b) t_i is either a linear constructor term or a ground $\mathcal{R}_u$-irreducible term [7, Definition 7.4.8], where $\mathcal{R}_u$ is obtained from the rules of $\mathcal{R}$ by just dropping the conditional part: $\mathcal{R}_u = \{\ell \to r \mid \ell \to r \Leftarrow c \in \mathcal{R}\}$. [7, Definition 7.1.2].

  By [7, Corollary 7.4.11], almost orthogonal and almost normal (i.e., right-stable and oriented [7, Definition 7.4.8(2)]) 2-CTRSs are confluent.

Thus, we let (using the fact that confluence implies local confluence):

$$\mathsf{P}_{\mathit{Orth}}(\mathit{CR}(\mathcal{R})) = \mathsf{P}_{\mathit{Orth}}(\mathit{WCR}(\mathcal{R})) = \emptyset \text{ if } \mathcal{R} \text{ is } \begin{cases} \text{a weakly orthogonal TRS, or} \\ \text{a } \mu\text{-orthogonal CS-TRS, or} \\ \text{an almost orthogonal and almost normal} \\ \quad \text{2-CTRS, or} \\ \text{an orthogonal, properly oriented, and right-stable} \\ \quad \text{3-CTRS} \end{cases}$$

In all these uses, $\mathsf{P}_{\mathit{Orth}}$ is *sound* and (trivially) *complete*.

Example 5.23. (Continuing Example 5.8)
Since the TRS $\mathcal{R}_1$ in Example 5.8, is orthogonal, we obtain $\mathsf{P}_{\mathit{Orth}}(\mathit{CR}(\mathcal{R}_1)) = \emptyset$.

5.4.4. Confluence and local/strong confluence: P_{CR}, $\mathsf{P}_{\mathit{WCR}}$, $\mathsf{P}_{\mathit{SCR}}$

Processors P_{CR}, $\mathsf{P}_{\mathit{WCR}}$ and $\mathsf{P}_{\mathit{SCR}}$ implement the well-known relationships between confluence and local and strong confluence in Remark 4.3 from which soundness and completeness properties of these processors follow.

Local and strong confluence problems as confluence problems. The following processor transforms local confluence problems into confluence problems: for GTRSs $\mathcal{R}$,

$$\mathsf{P}_{\mathit{CR}}(\mathit{WCR}(\mathcal{R})) = \mathsf{P}_{\mathit{CR}}(\mathit{SCR}(\mathcal{R})) = \{\mathit{CR}(\mathcal{R})\}.$$

P_{CR} is *sound* on local confluence problems, and *complete* on strong confluence problems.

[6]Originally in [43, Theorem 4.6], this result concerns *level-confluence*, which implies confluence.

Confluence and strong confluence problems as local confluence problems. The following processors transforms confluence problems into local confluence problems: for GTRSs $\mathcal{R}$.

$$\mathsf{P}_{WCR}(CR(\mathcal{R})) = \mathsf{P}_{WCR}(SCR(\mathcal{R})) = \{WCR(\mathcal{R})\}.$$

P_{WCR} is *complete*, but not *sound.*

Confluence and local confluence problems as strong confluence problems. Strong confluence implies confluence and hence local confluence (but not vice versa) [37, Lemma 2.5], see also [7, Section 2.2]. The following processor uses this fact: for GTRSs $\mathcal{R}$,

$$\mathsf{P}_{SCR}(CR(\mathcal{R})) = \mathsf{P}_{SCR}(WCR(\mathcal{R})) = \{SCR(\mathcal{R})\}.$$

P_{SCR} is *sound*, but *not* complete.

5.4.5. Confluence of a TRS as local confluence of a terminating CS-TRS: P_{CanJ}, P_{CnvJ}

Replacement maps can be used to prove confluence of a TRS $\mathcal{R}$ by transforming it the into a CS-TRS $(\mathcal{R}, \mu)$. We consider two replacement maps:

- The *canonical replacement map* $\mu^{can}_{\mathcal{R}}$ is *the most restrictive replacement map ensuring that the non-variable subterms of the left-hand sides of the rules of* $\mathcal{R}$ *are all active* [2, Section 5].
- The *convective replacement map* $\mu^{cnv}_{\mathcal{R}}$ is *the most restrictive replacement map that makes all critical positions* p *of critical pairs* $\langle\theta(\ell)[\theta(r')]_p, \theta(r)\rangle \in \mathsf{CP}(\mathcal{R})$ *active* [38, Definition 3].

Replacement maps μ that are *less restrictive* than $\mu^{can}_{\mathcal{R}}$ (i.e., such that $\mu^{can}_{\mathcal{R}}(f) \subseteq \mu(f)$ for all symbols f in the signature, written $\mu^{can}_{\mathcal{R}} \sqsubseteq \mu$) are collected in the set $CM_{\mathcal{R}}$ and similarly for $\mu^{cnv}_{\mathcal{R}}$ with $CnvM_{\mathcal{R}}$. Since $\mu^{can}_{\mathcal{R}}$ is not more restrictive than $\mu^{cnv}_{\mathcal{R}}$, i.e., $\mu^{cnv}_{\mathcal{R}} \sqsubseteq \mu^{can}_{\mathcal{R}}$, we have $CM_{\mathcal{R}} \subseteq CnvM_{\mathcal{R}}$.

Example 5.24. Consider the following TRS (COPS/42.trs):

$$\begin{aligned} \mathsf{f}(\mathsf{g}(x)) &\to \mathsf{f}(\mathsf{h}(x,x)) && (39)\\ \mathsf{g}(\mathsf{a}) &\to \mathsf{g}(\mathsf{g}(\mathsf{a})) && (40)\\ \mathsf{h}(\mathsf{a},\mathsf{a}) &\to \mathsf{g}(\mathsf{g}(\mathsf{a})) && (41) \end{aligned}$$

We have $\mu^{can}_{\mathcal{R}} = \mu_\top$. The critical position $1 \in \mathcal{P}os(\ell_{(39)})$ of the only critical pair $\langle \mathsf{f}(\mathsf{g}(\mathsf{g}(\mathsf{a}))), \mathsf{f}(\mathsf{h}(\mathsf{a},\mathsf{a}))\rangle$ becomes *active* by just letting $\mu^{cnv}_{\mathcal{R}}(\mathsf{f}) = \{1\}$ and $\mu^{cnv}_{\mathcal{R}}(p) = \emptyset$ for any other symbol p.

Example 5.25. Consider the TRS $\mathcal{R}$ in Example 5.8.We have:

$$\mu^{can}_{\mathcal{R}}(\mathsf{inc}) = \mu^{can}_{\mathcal{R}}(\mathsf{hd}) = \mu^{can}_{\mathcal{R}}(\mathsf{tl}) = \{1\}, \quad \mu^{can}_{\mathcal{R}}(:) = \mu^{can}_{\mathcal{R}}(\mathsf{from}) = \mu^{can}_{\mathcal{R}}(\mathsf{s}) = \emptyset.$$

Also, with rule (26), i.e., $\mathsf{inc}(\mathsf{tl}(\mathsf{from}(x))) \to \mathsf{tl}(\mathsf{inc}(\mathsf{from}(x)))$, critical position $p = 1.1$ in the left-hand side $\mathsf{inc}(\mathsf{tl}(\mathsf{from}(x)))$ of the rule, and rule (25), i.e., $\mathsf{from}(x) \to x{:}\mathsf{from}(\mathsf{s}(x))$, we obtain a critical pair

$$\langle \mathsf{inc}(\mathsf{tl}(x{:}\mathsf{from}(\mathsf{s}(x)))), \mathsf{tl}(\mathsf{inc}(\mathsf{from}(x)))\rangle \qquad (42)$$

which is the only critical pair in $\mathsf{CP}(\mathcal{R})$. We only need to make the arguments of inc and tl active to guarantee that p is active in $\mathsf{inc}(\mathsf{tl}(\mathsf{from}(x)))$. Thus, we have:

$$\mu^{cnv}_{\mathcal{R}}(\mathsf{inc}) = \mu^{cnv}_{\mathcal{R}}(\mathsf{tl}) = \{1\}, \quad \mu^{cnv}_{\mathcal{R}}(:) = \mu^{cnv}_{\mathcal{R}}(\mathsf{from}) = \mu^{cnv}_{\mathcal{R}}(\mathsf{hd}) = \mu^{cnv}_{\mathcal{R}}(\mathsf{s}) = \emptyset.$$

A CS-TRS $(\mathcal{R}, \mu)$ is

- *level-decreasing* if for all rules $\ell \to r$ in $\mathcal{R}$, the level of each variable in r does not exceed its level in ℓ; the *level* $lv_\mu(t, p)$ of an occurrence of variable x at position p in a term t is obtained by adding the number of frozen arguments that are traversed from the root to the occurrence $t|_p = x$ of the variable. Then, $lv_\mu(t, x)$ is the maximum level to which x occurs in t [44, Definition 1].
- 0-*preserving* [38, page 6] if for all rules $\ell \to r \in \mathcal{R}$, if a variable occurs active in the left-hand side ℓ of rules $\ell \to r$, then all its occurrences in r are also active, i.e., $\mathcal{V}ar^{\mu}(\ell) \cap \mathcal{V}ar^{\not\mu}(r) = \emptyset$. For left-linear TRSs, this property coincides with $\ell \to r$ having *left-homogeneous μ-replacing variables* [2, Section 8.1], written $\mathsf{LHRV}(\mathcal{R}, \mu)$, see, e.g., [25, Proposition 3].

Example 5.26. Consider the TRS $\mathcal{R}$ in Example 5.8.

- With $\mu = \mu^{can}_{\mathcal{R}}$, $\mathcal{R}$ is *not* level-decreasing, as for rule (25),

$$lv_\mu(\ell_{(25)}, x) = lv_\mu(\mathsf{from}(x), x) = 0 < 2 = lv_\mu(x{:}\mathsf{from}(\mathsf{s}(x)), x) = lv_\mu(r_{(25)}, x).$$

- With respect to both $\mu^{cnv}_{\mathcal{R}}$ and $\mu^{can}_{\mathcal{R}}$, $\mathcal{R}$ is 0-preserving as all variable occurrences in left-hand sides of rules are frozen.

Let $\mathcal{R} = (\mathcal{F}, R)$ be a left-linear TRS and assume $\mathcal{R}^\mu = (\mathcal{F}, \mu, R)$ terminating for μ as given below. We define

$$\begin{array}{rcll} \mathsf{P}_{CanJ}(CR(\mathcal{R})) & = & \{WCR(\mathcal{R}^\mu)\} & \text{if } \mu \in CM_{\mathcal{R}} \text{ and } \mathcal{R} \text{ is level-decreasing.} \\ \mathsf{P}_{CnvJ}(CR(\mathcal{R})) & = & \{WCR(\mathcal{R}^\mu)\} & \text{if } \mu \in CnvM_{\mathcal{R}} \text{ and } \mathsf{LHRV}(\mathcal{R}, \mu) \text{ holds.} \end{array}$$

Example 5.27. (Continuing Example 5.8)
Since $\mathcal{R}_2$ in Example 5.8 is left-linear and $\mathsf{LHRV}(\mathcal{R}_2, \mu^{cnv}_{\mathcal{R}_2})$ holds, we obtain $\mathsf{P}_{CnvJ}(CR(\mathcal{R}_2)) = \{WCR(\mathcal{R}^\mu_2)\}$, where $\mathcal{R}^\mu_2 = (\mathcal{R}_2, \mu^{cnv}_{\mathcal{R}_2})$.

By [44, Theorem 2] (resp. [38, Corollary 13]), P_{CanJ} (resp. P_{CnvJ}) is *sound*. However, they are *not* complete.

Example 5.28. (P_{CanJ} and P_{CnvJ} are not complete)
For $\mathcal{R}$ consisting of the rules

$$\mathsf{a} \to \mathsf{b} \tag{43}$$

$$\mathsf{c} \to \mathsf{d}(\mathsf{a}) \tag{44}$$

$$\mathsf{c} \to \mathsf{d}(\mathsf{b}) \tag{45}$$

we have $\mu^{can}_{\mathcal{R}} = \mu^{cnv}_{\mathcal{R}} = \mu_\bot$. In particular, $\mu^{cnv}_{\mathcal{R}}(\mathsf{d}) = \emptyset$. The (only) critical pair

$$\langle \mathsf{d(a)}, \mathsf{d(b)} \rangle \tag{46}$$

is *not* $(\mathcal{R}, \mu_\bot)$-joinable, hence $(\mathcal{R}, \mu_\bot)$ is *not* locally confluent. However, $\mathcal{R}$ is confluent as it is terminating and π is clearly $\mathcal{R}$-joinable.

Remark 5.29. **(P_{CnvJ} subsumes P_{CanJ})**
Since $\mu^{cnv}_{\mathcal{R}} \sqsubseteq \mu^{can}_{\mathcal{R}}$, and level-decreasingness implies the LHRV property (see [44, page 72]), and hence 0-decreasingness, all uses of P_{CanJ} are covered by P_{CnvJ}. In practice, though, this may depend on the specific choice(s) of μ when implementing the processor. For instance, consider $\mathcal{R}'$ consisting of rules (43), (44), and (45), together with

$$\mathsf{d(b)} \ \to \mathsf{b} \tag{47}$$

For $\mathcal{R}'$ and $\mathcal{R}$ in Example 5.28, with $\mathsf{CP}(\mathcal{R}) = \mathsf{CP}(\mathcal{R}') = \{(46)\}$, we have $\mu^{cnv}_{\mathcal{R}} = \mu^{cnv}_{\mathcal{R}'} = \mu_\bot$. Still, (46) is not $(\mathcal{R}', \mu_\bot)$-joinable. However, $\mu^{can}_{\mathcal{R}'}(\mathsf{d}) = \{1\}$ now (due to rule (47)) and hence (46) is $(\mathcal{R}', \mu^{can}_{\mathcal{R}'})$-joinable. Thus, $(\mathcal{R}', \mu^{can}_{\mathcal{R}'})$ is locally confluent and (by soundness of P_{CanJ} and termination of $(\mathcal{R}', \mu^{can}_{\mathcal{R}'})$), $\mathcal{R}$ is confluent. Typically, an implementation of P_{CnvJ} would try $\mu^{cnv}_{\mathcal{R}}$ *only*, although $\mu^{can}_{\mathcal{R}} \in \mathit{CnvM}_{\mathcal{R}}$ is a possible choice as well.

5.4.6. Confluence of a TRS as confluence of canonical CSR: P_{CanCR}

By relying on [2, Corollary 8.23], processor P_{CanCR} transforms a confluence problem $\mathit{CR}(\mathcal{R})$ for a TRS $\mathcal{R}$ into a confluence problem for a CS-TRS:

$$\mathsf{P}_{CanCR}(\mathit{CR}(\mathcal{R})) = \{\mathit{CR}(\mathcal{R}^\mu)\}$$

if $\mathcal{R}$ is a left-linear and *normalizing* TRS (i.e., every term has a normal form), and $\mathcal{R}^\mu = (\mathcal{R}, \mu)$ for some $\mu \in \mathit{CM}_{\mathcal{R}}$. P_{CanCR} is *sound* but *not* complete, see [2, Sections 8.4 & 8.5] for a discussion.

5.4.7. Confluence of terminating systems as local confluence: P_{KB}

The characterization of confluence of terminating TRSs as the joinability of all its critical pairs is a landmark, early result by Knuth and Bendix [45]. Thus, we let

$$\mathsf{P}_{KB}(\mathit{CR}(\mathcal{R})) = \begin{cases} \{\mathit{JO}(\mathcal{R}, \pi_1), \ldots, \mathit{JO}(\mathcal{R}, \pi_n)\} & \text{if } \{\pi_1, \ldots, \pi_n\} = \mathsf{pCCP}(\mathcal{R}) \cup \mathsf{iCCP}(\mathcal{R}) \\ & \text{are overlays and } \mathcal{R} \text{ is a J-CTRS} \\ \{\mathit{WCR}(\mathcal{R})\} & \text{otherwise} \end{cases}$$

if $\mathcal{R}$ is a terminating GTRS [15]. By relying on [46, Theorem 4] for the first case of the application of P_{KB} and Theorem 5.9 (plus Newman's Lemma) for the second one, P_{KB} is *sound* and *complete*.

Example 5.30. (Continuing Example 5.1)
For the *oriented* 1-CTRS $\mathcal{R}'$ obtained by P_{Simp} in Example 5.1, there is a proper CCP:

$$\langle \mathsf{h}(\mathsf{s}(x)), \mathsf{g}(\mathsf{s}(x))\rangle \Leftarrow \mathsf{c}(\mathsf{g}(x)) \approx \mathsf{c}(\mathsf{a}), \mathsf{c}(\mathsf{h}(x)) \approx \mathsf{c}(\mathsf{a}) \quad (48)$$

which is an *overlay*, as the critical position is $p = \Lambda$. Since $\mathcal{R}'$ is a 1-CTRS, we dismiss improper critical pairs. Thus, we have $\mathsf{P}_{KB}(CR(\mathcal{R}')) = \{JO(\mathcal{R}', (48))\}$.

Example 5.31. (Continuing Examples 1.1 and 3.6)
The CS-CTRS $\mathcal{R}_\perp$ in Example 1.1, is clearly terminating. Thus, $\mathsf{P}_{KB}(CR(\mathcal{R}_\perp)) = \{WCR(\mathcal{R}_\perp)\}$.

5.5. Joinability processor: P_{JO}

For GTRSs $\mathcal{R}$ and conditional pairs π, we have the following processor:

$$\mathsf{P}_{JO}(JO(\mathcal{R},\pi)) = \begin{cases} \emptyset & \text{if } \pi \text{ is joinable} \\ \mathsf{no} & \text{otherwise} \end{cases}$$

$$\mathsf{P}_{JO}(SJO(\mathcal{R},\pi)) = \begin{cases} \emptyset & \text{if } \pi \text{ is strongly joinable} \\ \mathsf{no} & \text{otherwise} \end{cases}$$

For both uses P_{JO} is *sound* and *complete*. As in [22, 1, 25], we often prove (non)joinability of terms and critical pairs by proving the *(in)feasibility* of sequences (see Section 2).

Proposition 5.32. (cf. [1, Proposition 21 & Section 7.5]) Let $\mathcal{R}$ be a GTRS and $\pi : \langle s,t\rangle \Leftarrow c$ be a conditional pair with variables $\vec{x}$, and z be a variable not in $\vec{x}$. If (i) $\sigma(c)$ is feasible for some substitution σ, and (ii) $\sigma(c), \sigma(s) \to^* z, \sigma(t) \to^* z$ is infeasible (for some $z \notin \mathcal{V}ar(\sigma(c),\sigma(s),\sigma(t))$), then π is not joinable.

Remark 5.33. As discussed in [1, Remark 22], in order to use Proposition 5.32, the following *heuristics* are useful.

H1 The simplest choice is the empty substitution, i.e., $\sigma = \varepsilon$ or its *grounded* version $\sigma = \varepsilon^{\downarrow} = \{x \mapsto c_x \mid x \in \mathcal{X}\}$. This is easily mechanizable, see Examples 5.34 and 5.37.

H2 Use $\sigma = \{x \mapsto \ell^{\downarrow}, x' \mapsto r^{\downarrow}\}$ for some unconditional rule $\ell \to r$ in $\mathcal{R}$ if π is a conditional variable pair $\langle s,t\rangle \Leftarrow x \to x', c$.

H3 Choosing another substitution σ, usually trying to fulfill the conditions in the proposition.

Example 5.34. (Continuing Example 1.1)
We use Proposition 5.32 to show non-joinability of (6): the sequence $\mathsf{s}(x') \to^* \mathsf{s}(0)$ is feasible, but

$$\mathsf{s}(x') \to^* \mathsf{s}(0), \mathsf{f}(\mathsf{g}(x')) \to^* z, \mathsf{s}(x') \to^* z$$

is infeasible: the only substitution satisfying $\mathsf{s}(x') \to^* \mathsf{s}(0)$ is $\sigma = \{x' \mapsto 0\}$; furthermore, in order to satisfy the last condition $\mathsf{s}(x') \to^* z$ we need $\sigma = \{x' \mapsto 0, z \mapsto 0\}$; however, $\sigma(\mathsf{f}(\mathsf{g}(x'))) = \mathsf{f}(\mathsf{g}(0))$

is irreducible; thus $\mathsf{f}(\mathsf{g}(\mathsf{0})) \to^* \mathsf{0}$ is *not* satisfied. By Proposition 5.32, the conditional critical pair (6) is not joinable. By Theorem 5.9.(1), $\mathcal{R}$ in Example 1.1 is not locally $\overline{\mathcal{R}}$-confluent nor $\overline{\mathcal{R}}$-confluent. As for $\mathcal{R}_\perp$, since $\mathsf{pCCP}(\mathcal{R}_\perp) = \mathsf{iCCP}(\mathcal{R}_\perp) = \mathsf{CVP}(\mathcal{R}_\perp) = \emptyset$ (see Example 3.6), by Theorem 5.9.(1), $\mathcal{R}_\perp$ is locally confluent. Since $\mathcal{R}_\perp$ is terminating, by Newman's Lemma, $\mathcal{R}_\perp$ is confluent.

The following results, originally established in [22] for CTRSs, can also be used with GTRSs.

Proposition 5.35. Let $\mathcal{R}$ be a GTRS and $\pi : \langle s, t\rangle \Leftarrow c$ be a conditional pair with variables $\vec{x}$, and z be a variable not in $\vec{x}$. Then,

1. (cf. [22, Corollary 17]) If $\overline{\mathcal{R}} \vdash (\forall \vec{x})(\exists z)\; c \Rightarrow s \to^* z \wedge t \to^* z$ holds, then π is joinable.
2. (cf. [22, Corollary 18]) If $s^{\downarrow} \to^* z \wedge t^{\downarrow} \to^* z$ is feasible, then π is joinable.
3. (cf. [22, Proposition 22]) If $\mathcal{V}ar(c) \cap \mathcal{V}ar(s,t) = \emptyset$, then π is joinable iff c is infeasible or $s^{\downarrow} \to^* z, t^{\downarrow} \to^* z$ is feasible.

The tool infChecker [16] can be used to automatically prove (in)feasibility of sequences.

Example 5.36. (Continuing Example 5.27)
The (unconditional) critical pair (42), i.e., $\langle \mathsf{inc}(\mathsf{tl}(x{:}\mathsf{from}(\mathsf{s}(x)))), \mathsf{tl}(\mathsf{inc}(\mathsf{from}(x)))\rangle$, obtained in Example 5.27 for $\mathcal{R}_2$ in Example 5.8 is $\overline{\mathcal{R}}^{\mu}_2$-joinable:

$$\mathsf{inc}(\underline{\mathsf{tl}(x{:}\mathsf{from}(\mathsf{s}(x)))}) \quad \to_{\mathcal{R}^{\mu}_2} \quad \mathsf{inc}(\mathsf{from}(\mathsf{s}(x)))$$

and

$$\mathsf{tl}(\mathsf{inc}(\underline{\mathsf{from}(x)})) \to_{\mathcal{R}^{\mu}_2} \mathsf{tl}(\underline{\mathsf{inc}(x{:}\mathsf{from}(\mathsf{s}(x)))}) \to_{\mathcal{R}^{\mu}_2} \underline{\mathsf{tl}(\mathsf{s}(x){:}\mathsf{inc}(\mathsf{from}(\mathsf{s}(x))))} \to_{\mathcal{R}^{\mu}_2} \mathsf{inc}(\mathsf{from}(\mathsf{s}(x))).$$

Thus, we have $\mathsf{P}_{JO}(JO(\mathcal{R}^{\mu}_2, (42))) = \emptyset$.

Example 5.37. (Continuing Example 3.5)
Consider the O-CTRS $\mathcal{R}$ in Example 1.1 and the conditional critical pair (6), i.e., $\langle \mathsf{f}(\mathsf{g}(x')), \mathsf{s}(x')\rangle \Leftarrow \mathsf{s}(x') \approx \mathsf{s}(\mathsf{0})$. This conditional pair is *not* joinable as the only way to satisfy the reachability condition $\mathsf{s}(x') \approx \mathsf{s}(\mathsf{0})$ in the conditional part is using $\sigma = \{x' \mapsto \mathsf{0}\}$. However, $\sigma(\mathsf{f}(\mathsf{g}(x'))) = \mathsf{f}(\mathsf{g}(\mathsf{0}))$, and $\sigma(\mathsf{s}(x')) = \mathsf{s}(\mathsf{0})$ are both irreducible. Alternatively, using Proposition 5.32 and heuristic H1 in Remark 5.33, the non-joinability of (6) can be proved as the $\overline{\mathcal{R}}$-*infeasibility* of the sequence

$$\mathsf{s}(x') \to^* \mathsf{s}(\mathsf{0}), \mathsf{f}(\mathsf{g}(x')) \to^* z, \mathsf{s}(x') \to^* z \tag{49}$$

where z is a fresh variable. This can be proved by infChecker (use the input in Figure 2).

Remark 5.38. (Simple methods for joinability)
The previous methods, which are based on translating joinability proofs into (in)feasibility proofs, heavily rely on the use of infChecker to solve them. Since this is costly, some exploration techniques for concluding joinability are used. For *unconditional* pairs $\pi : \langle s, t\rangle$,

```
(PROBLEM INFEASIBILITY)
(CONDITIONTYPE ORIENTED)
(VAR x)
(RULES
  f(g(x)) -> x | x == s(0)
  g(s(x)) -> g(x)
)
(VAR x' z)
(CONDITION s(x') ->* s(0), f(g(x')) ->* z, s(x') ->* z
)
```

Figure 2. Non-joinability as infeasibility in Example 5.37

J0 If s and t are *irreducible* and $s \neq t$, then π is *not* joinable.

Given an unconditional pair $\pi : \langle s, t\rangle$ or a feasible conditional pair $\pi : \langle s, t\rangle \Leftarrow c$,

J1 If s and t are both ground and *irreducible* and $s \neq t$, then π is *not* joinable.

J2 If $\mathsf{Suc}_{\to_{\mathcal{R}}}(s) \cap \mathsf{Suc}_{\to_{\mathcal{R}}}(t) \neq \emptyset$, then π is joinable.

J3 If $\mathsf{Suc}^{=}_{\to_{\mathcal{R}}}(t) \cap \mathsf{Suc}_{\to_{\mathcal{R}}}(s) \neq \emptyset$ and $\mathsf{Suc}^{=}_{\to_{\mathcal{R}}}(s) \cap \mathsf{Suc}_{\to_{\mathcal{R}}}(t) \neq \emptyset$, then π is strongly joinable.

Remark 5.39. (Checking irreducibility in J0 and J1)
Dealing with TRSs $\mathcal{R}$, we can check *irreducibility* of a term u by just showing that u contains *no redex*, i.e., no instance $\sigma(\ell)$ of a left-hand side ℓ of a rule $\ell \to r$ in $\mathcal{R}$ occurs in u. Dealing with GTRSs $\mathcal{R}$, this simple test is not enough: there can be terms including instances $\sigma(\ell)$ of the left-hand side ℓ of a *conditional* rule $\ell \to r \Leftarrow c$ (they are called *preredexes* [47]), which are also *irreducible* if $\overline{\mathcal{R}} \vdash \sigma(c)$ does *not* hold. This is implemented by using a model generator (e.g., AGES [48] or Mace4 [49]) to (try to) find a *countermodel* of $(\forall\vec{x})\sigma(c)$ in $\overline{\mathcal{R}}$, where $\vec{x}$ are the variables occurring in $\sigma(c)$. We proceed in two steps:

1. If u contains no (pre)redex, then it is irreducible.
2. If u contains a preredex $\sigma(\ell)$ of a conditional rule $\ell \to r \Leftarrow A_1, \ldots, A_n$ and the theory

$$\overline{\mathcal{R}} \cup \{\neg(\forall\vec{x})\ \sigma(A_1) \wedge \cdots \wedge \sigma(A_n)\},$$

for $\vec{x}$ the variables occurring in $\sigma(A_1), \cdots, \sigma(A_n)$ is *satisfiable*, then u is irreducible.

Example 5.40. (Continuing Example 5.30)
The proper CCP (48), i.e., $\langle \mathsf{h}(\mathsf{s}(x)), \mathsf{g}(\mathsf{s}(x))\rangle \Leftarrow \mathsf{c}(\mathsf{g}(x)) \approx \mathsf{c}(\mathsf{a}), \mathsf{c}(\mathsf{h}(x)) \approx \mathsf{c}(\mathsf{a})$, is joinable, as we have $\mathsf{h}(\mathsf{s}(x)) \to_{(10)} x$ and $\mathsf{g}(\mathsf{s}(x)) \to_{(9)} x$. Thus, $\mathsf{P}_{JO}(JO(\mathcal{R}', (48))) = \emptyset$.

Figures 3 and 4 display the proof trees of the confluence framework for the examples in the paper.

$$\begin{tikzcd}
& CR(\mathcal{R}) \arrow[d, "\mathsf{P}_{WCR}"] & \\
& WCR(\mathcal{R}) \arrow[dl] \arrow[dr] \arrow[d, phantom, "\mathsf{P}_{HE}"] & \\
JO(\mathcal{R},(6)) \arrow[d, "\mathsf{P}_{JO}"] & & JO(\mathcal{R},(7)) \\
\textsf{no} & &
\end{tikzcd}$$

CTRS $\mathcal{R}$ in Example 1.1;
see also Examples 3.5, 5.10, and 5.34

$$\begin{tikzcd}
CR(\mathcal{R}_\perp) \arrow[d, "\mathsf{P}_{KB}"] \\
WCR(\mathcal{R}_\perp) \arrow[d, "\mathsf{P}_{HE}"] \\
\textsf{yes}
\end{tikzcd}$$

CS-CTRS $\mathcal{R}_\perp$ in Example 1.1;
see also Examples 3.6 and 5.31

Figure 3. Proof trees for confluence problems of CTRSs and CS-CTRSs in the confluence framework

$$\begin{tikzcd}
CR(\mathcal{R}) \arrow[d, "\mathsf{P}_{Simp}"] \\
CR(\mathcal{R}') \arrow[d, "\mathsf{P}_{KB}"] \\
JO(\mathcal{R}',(48)) \arrow[d, "\mathsf{P}_{JO}"] \\
\textsf{yes}
\end{tikzcd}$$

CTRS $\mathcal{R}$ in Example 5.1;
see also Examples 5.30 and 5.40

$$\begin{tikzcd}
& CR(\mathcal{R}) \arrow[dl] \arrow[dr] \arrow[d, phantom, "\mathsf{P}_{MD}"] & \\
CR(\mathcal{R}_1) \arrow[d, "\mathsf{P}_{Orth}"] & & CR(\mathcal{R}_2) \arrow[d, "\mathsf{P}_{CnvJ}"] \\
\textsf{yes} & & WCR(\mathcal{R}_2^\mu) \arrow[d, "\mathsf{P}_{HE}"] \\
& & JO(\mathcal{R}_2^\mu,(42)) \arrow[d, "\mathsf{P}_{JO}"] \\
& & \textsf{yes}
\end{tikzcd}$$

TRS $\mathcal{R}$ in Example 5.8;
see also Examples 5.23, 5.27, and 5.36

Figure 4. Proof trees for confluence problems of CTRSs and TRSs in the confluence framework

6. Strategy

CONFident implements several processors. Given a GTRS $\mathcal{R}$, the processors enumerated in Section 5 are used to build a proof tree with root $CR(\mathcal{R})$ to hopefully conclude confluence or non-confluence of $\mathcal{R}$ (Theorem 4.6). The selection and combination of processors to generate such a proof tree is usually encoded as a fixed *proof strategy* which is applied to the initial confluence problem $CR(\mathcal{R})$. Choosing the appropriate proof strategy for an input problem is not a trivial task. The strategy must take into account that many proof obligations involved in the implementation of processors are *undecidable*. For instance, joinability of conditional pairs (required, e.g., by P_{JO}) is, in general, undecidable. Also, calls to external tools trying to check undecidable properties like termination (as required, e.g., by P_{KB}) may fail or succeed in proving the *opposite* property, i.e., non-termination. For these reasons, the application of processors is usually constrained by a *timeout* so that after a predefined amount of time, the strategy may try a different processor hopefully succeeding on the considered problem, or even *backtrack* to a previous problem in the proof tree. Typically, a thorough experimental analysis is needed to obtain a suitable strategy.

The proof strategies for (CS-)TRSs and (CS-)CTRSs are depicted in Figures 5 and 6, respectively. The diagrams show how to deal with CR, WCR, SCR, JO, and SJO problems for (CS-)TRSs and (CS-)CTRSs in CONFident, according to currently implemented techniques. Starting from the box identifying the problem at stake, the sequence of used processors is displayed and the order of application is indicated by means of arrows from one processor to the next one. Some processors have two possible continuations. Some of them correspond to *ending* applications of the processor leading to finish some branch (represented as boxes enclosing yes or no) and the decision depends on the result of the processor as indicated as a label in the branches (e.g., for P_{HE}, P_{Orth}, P_{KB}, P_{JO}). In other cases, the continuous one is chosen first and the *dashed* one is followed after a *failure* in the first option.

Example 6.1. (Use of P_{KB})
For (CS-)TRSs $\mathcal{R}$ (see Figure 5), if $\mathcal{R}$ is terminating, then P_{KB} produces a call to $\mathsf{P}_{HE}(\mathit{WCR}(\mathcal{R}))$. Otherwise, P_{CnvJ} is used. For (CS-)CTRSs (see Figure 6), if $\mathcal{R}$ is terminating, then

1. If $\mathcal{R}$ is a J-CTRS, then, if $\mathsf{pCCP}(\mathcal{R}) \cup \mathsf{iCCP}(\mathcal{R}) = \emptyset$, then a positive answer yes is given; otherwise, if all pairs in $\mathsf{pCCP}(\mathcal{R}) \cup \mathsf{iCCP}(\mathcal{R})$ are overlays, then calls to P_{JO} for each $\pi \in \mathsf{pCCP}(\mathcal{R}) \cup \mathsf{iCCP}(\mathcal{R})$ are made.

2. If $\mathcal{R}$ is not a J-CTRS, or $\mathsf{pCCP}(\mathcal{R}) \cup \mathsf{iCCP}(\mathcal{R})$ is not a set of overlays, then P_{KB} produces a call to $\mathsf{P}_{HE}(\mathit{WCR}(\mathcal{R}))$.

Otherwise, P_{WCR} is called.

Note that, in proofs involving (CS-)CTRSs $\mathcal{R}$, after applying P_{Inl} and P_{Simp}, it is possible to obtain a (CS-)TRS $\mathcal{R}'$. In this case, the proof would continue in the corresponding point of Figure 5.

7. Structure of CONFident

CONFident is written in Haskell and it has more than 100 Haskell files with almost 15000 lines of code (blanks and comments not included). The tool is used online through its web interface in:

`http://zenon.dsic.upv.es/confident/`

Nowadays, only *confluence problems* $\mathit{CR}(\mathcal{R})$ for CS-CTRS*s* $\mathcal{R}$ can be (explicitly) solved, i.e., the input system $\mathcal{R}$ is treated as a CS-CTRS and a proof of *confluence* of $\mathcal{R}$ is attempted. Direct proofs of local or strong confluence, or (strong) joinability are not possible yet.

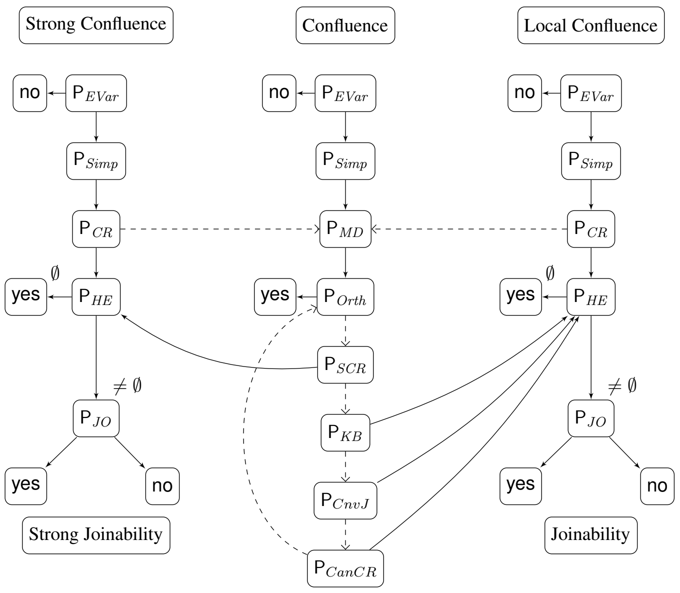

Figure 5. Proof strategy for (CS-)TRSs

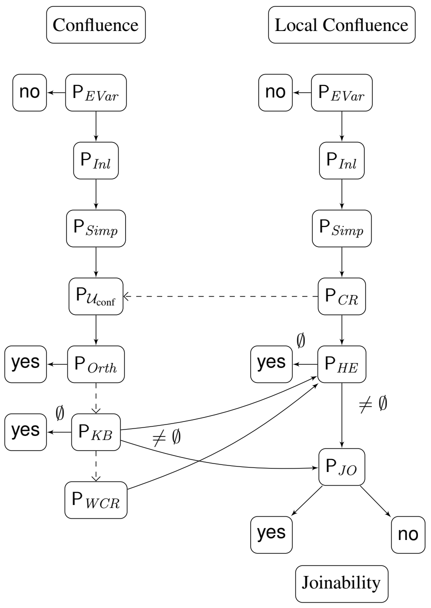

Figure 6. Proof strategy for (CS-)CTRSs

```
(CONDITIONTYPE ORIENTED)
(REPLACEMENT-MAP
  (f )
  (g )
  (s )
)
(VAR x)
(RULES
  g(s(x)) -> g(x)
  f(g(x)) -> x | x == s(0)
)
```

```
(STRATEGY CONTEXTSENSITIVE
  (f )
  (g )
  (s )
)
(VAR x)
(RULES
  g(s(x)) -> g(x)
  f(g(x)) -> x | x ->* s(0)
)
```

Figure 7. The CS-CTRS $\mathcal{R}_\perp$ in Example 1.1 in COPS (left) and TPDB (right) format

7.1. Input format

The main input format to introduce rewrite systems in CONFident is the COPS format,[7] the official input format of the Confluence Competition (CoCo[8]). The (older) TPDB format[9] can also be used. As an example, Figure 7 displays the COPS and TPDB encoding of the O-CS-CTRS $\mathcal{R}_\perp$ in Example 1.1. Both formats provide a similar display of the example, as they organize the information about the rewrite system in several blocks: the type of CTRS according to the evaluation of conditions (only in the COPS format, as the TPDB format permits *oriented* CTRSs only); replacement map specification (within a section `REPLACEMENT-MAP` in the COPS format and within a section `STRATEGY CONTEXTSENSITIVE` in the TPDB format); variable declaration; and rule description.

In CONFident, the user can combine both formats if needed (avoiding inconsistencies, of course), for example, using specific rewriting relations in conditions in the COPS format or using the reserved word `REPLACEMENT-MAP` to define a replacement map instead of the `STRATEGY` block in the TPDB format.

7.2. Use of external tools

CONFident uses specialized tools to solve auxiliary proof obligations. For instance,

- infChecker is used by P_{Simp} to prove infeasibility of conditional rules which are then discarded from the analysis. It is also used by P_{HE}, P_{Orth}, and P_{KB} to remove infeasible conditional critical pairs. Finally, infChecker is also used to implement the tests of joinability and μ-joinability of (conditional) pairs required by P_{JO}. For this purpose, Mace4 [49] and AGES [48] are used from infChecker.
- MU-TERM is used by P_{KB} and P_{CnvJ} to check termination of CTRSs and CS-TRSs.
- Prover9 [49] is used by P_{JO} to prove joinability of conditional pairs.

[7]http://project-coco.uibk.ac.at/problems/
[8]http://project-coco.uibk.ac.at/
[9]https://www.lri.fr/~marche/tpdb/

- Fort [50] is used by P_{CanCR} to check whether a TRS is normalizing.

Tools like MU-TERM or infChecker are connected as Haskell libraries that are directly used by CONFident, the rest of the tools are used by capturing external calls.

7.3. Implementing the confluence framework in Haskell

Problems. To implement the confluence framework, we start with the notion of problem. From the implementation point of view, a problem is just a data structure containing all the needed information to check its associated property. We can use a data type also to describe each kind of problem.

```haskell
-- | Problem
data Problem typ p = Problem typ p

-- | Confluence Problem Type
data ConfProblem = ConfProblem

-- | Joinability Problem Type
data JoinProblem = JoinProblem
```

However, because we want to give more flexibility and deal with variants of rewrite systems homogeneously, we prefer to define type classes to describe the common characteristics of each problem and define each variant of rewrite system as an proper instance.

```haskell
-- | Problems contains a rewrite system
class IsProblem typ problem | typ -> problem where
    getProblemType :: Problem typ problem -> typ

-- | Confluence Problems contains a rewrite system
class IsConfProblem typ problem trs | typ -> problem trs where
    getConfProblem :: Problem typ problem -> trs

-- | Joinability Problems have a list of critical pairs
class IsJoinProblem typ problem cp | typ -> problem cp where
    getCPair :: Problem typ problem -> cp
```

In these definitions, functional dependencies restrict a problem from being linked to multiple variants of rewrite systems and critical pairs. An example of instance can be CTRS:

```haskell
-- A CTRS is a Problem
instance IsProblem ConfProblem CR where
  getProblemType (Problem ConfProblem _) = ConfProblem

-- A CTRS is a Confluence Problem
instance IsConfProblem ConfProblem CR CR where
  getConfProblem (Problem ConfProblem trs) = trs
```

By using type classes, we can reuse methods among problems. In this case, a joinability problem is also a confluence problem because they share common methods.

```
-- A Pair (CR,CP) is a Problem
instance IsProblem JoinProblem JO where
  getProblemType (Problem JoinProblem _) = JoinProblem

-- A Conditional TRS and a (possibly conditional) Critical Pair is a Confluence Problem
instance IsConfProblem JoinProblem JO CR where
  getConfProblem (Problem JoinProblem jopair) = josystem jopair

-- A Conditional TRS and a (possible conditional) Critical Pair is a Joinability Problem
instance IsJoinProblem JoinProblem JO InCritPair where
  getCPair (Problem JoinProblem jopair) = joccpair jopair
```

Processors. After defining our problems, we need to find the way of defining processors in practice. The notion of processors is defined as abstractly as possible, allowing us to encapsulate every possible technique applied to a problem. Also, the implementation is kept as abstract as possible:

```
-- | Each processor is an instance of the class 'Processor'. The
-- output problem depends of the input problem and the applied processor
class Processor tag o d | tag o -> d where
apply        :: tag -> o -> Proof d
```

Each processor has its own name (`tag`). In the implementation, when a processor is applied, it yields a proof node. A proof contains the resulting set of problems or the refutation together with the infomation about the obtaind proof.

An example of proccessor can be the following dummy processor, a successful processor that returns a empty list of subproblems. Each processor returns information about its proof:

```
-- | Processor that returns an empty list
data SuccessProcessor = SuccessProcessor

-- | The information of the proof is just the input problem
data SuccessProcInfo problem = SuccessProcInfo { inSuccessProcInfoProblem :: problem }

-- | Returns an empty list of processors
instance Processor SuccessProcessor (Problem typ problem) (Problem typ problem) where
  apply SuccessProcessor inP
    = andP (SuccessProcInfo inP) inP []
```

The processor P_{HE} presented above accepts confluence problems as an input but returns joinability problems instead. Thus, the instances have the following form:

```
  instance Processor HuetProcessor (Problem ConfProblem problem1)
                                   (Problem JoinProblem problem2) where
   ...

  instance Processor CanCRProcessor (Problem ConfProblem problem1)
                                    (Problem ConfProblem problem2) where
   ...
```

Our framework is designed to be flexible enough to accommodate such processors.

We do not delve into implementation details of processors as they can vary significantly for each technique.

Proof trees. A proof node consists of the result of the applied processor in the form of an info data structure, the input problem, and a list of subproblems if the processor returns a set of subproblems. Consequently, we require a structure similar to the following:

```
-- | Proof Tree constructors
data ProofF k =
    And      { procInfo :: SomeProcInfo, problem :: SomeProblem, subProblems :: [k] }
```

```
  | Success { procInfo :: SomeProcInfo, problem :: SomeProblem }
  | Refuted { procInfo :: SomeProcInfo, problem :: SomeProblem }
  | DontKnow { procInfo :: SomeProcInfo, problem :: SomeProblem }
  | Search !([k])
```

`Success` is not necessary because it represents an `And` node with empty subproblems, but it aids us during the process of processing the solution. `DontKnow` is used to indicate a failure in the application of the processor, while `Search` is associated with the possibility of returning multiple solutions in the proof tree. `Search` is closely tied to how we define our strategies. It is important to note that Haskell utilizes lazy evaluation. As a result, we initially apply a strategy to the initial problem, and only when we traverse it to obtain the solution, the different processors are applied. Consequently, we may have a list of solutions, but ultimately we select the first successful one.

In our implementation, our proof is constructed by combining `ProofF` and `Problem` structures. By utilizing *free monads* [51], we can integrate the two types of nodes within the proof construction. The pure nodes (`Problem` instances) are represented as pure values encapsulated within the free monad, while the impure nodes (`ProofF` instances) are represented as impure effects within the free monad.

```
-- | 'Proof' is a Free Monad.
type Proof a = Free ProofF a
```

Because in our framework we have several kinds of problems and many types of processor answers, in the `ProofF` node we hide the type of problems using `SomeProblem` for the problems and `SomeProcInfo` for the processor answers. For this purpose, we use Generalized Algebraic Data Types (GADTs):

```
-- | 'SomeProblem' hides the type of the Problem
data SomeProblem where
    SomeProblem :: p -> SomeProblem

-- | 'SomeProcInfo' hides the type of the Process Output
data SomeProcInfo where
    SomeProcInfo :: p -> SomeProcInfo

-- | wraps the type of the problem
someProblem :: p -> SomeProblem
someProblem = SomeProblem

-- | wraps the type of the processor output
someProcInfo :: p -> SomeProcInfo
someProcInfo = SomeProcInfo
```

We define functions to simplify the returning of proofs by the processors:

```
-- | Return a success node
success :: procInfo -> problem1 -> Proof problem2
success pi p0 = Impure (Success (someProcInfo pi) (someProblem p0))

-- | Return a refuted node
```

```
refuted :: procInfo -> problem1 -> Proof problem2
refuted pi p0 = Impure (Refuted (someProcInfo pi) (someProblem p0))

-- | Return a dontKnow node
dontKnow :: procInfo -> problem1 -> Proof problem2
dontKnow pi p0 = Impure (DontKnow (someProcInfo pi) (someProblem p0))

-- | Return a list of subproblems
andP :: procInfo -> problem1 -> [problem2] -> Proof problem2
andP pi p0 [] = success pi p0
andP pi p0 pp = Impure (And (someProcInfo pi) (someProblem p0) (map return pp))
```

Strategies. Finally, to define proof strategies, we use two strategy combinators, the *sequential* combinator and the *alternative* combinator. This is implemented using the two functions[10]

```
-- | And strategy combinator
(.&.) :: (a -> Proof b) -> (b -> Proof c) -> a -> Proof c
(.&.) = (>=>)

-- | Or strategy combinator
(.|.) :: (a -> Proof b) -> (a -> Proof b) -> a -> Proof b
(f .|. g) p = (f p) `mplus` (g p)
```

In this section, we have introduced two types of problems: `ConfProblem` and `JoinProblem`. To enhance our strategy in practice, we can create new problem types that encapsulate desirable properties. By doing so, we can define specific strategies tailored to those problem types. This is particularly relevant when dealing with termination, where we can transition from `ConfProblem` to `TermConfProblem` and apply specialized strategies aimed at achieving finiteness.

The execution of our framework in the Main module can be summarized in the following three lines:

```
  let proof = crewstrat timeout problem
  let sol = runProof proof
  putStr . show . pPrint $ sol
```

where `csrewstrat` is a proof strategy of processors connected with combinators and `runProof` traverses the proof tree hopefully obtaining the solution. In our case, we follow a Breadth-First Search strategy, but other alternatives can be implemented.

8. Experimental results

CONFident participated in the 2022[11] and 2023[12] International Confluence Competition (CoCo) in the categories TRS, SRS, CSR, and CTRS. Table 7 summarizes the results obtained by the tool in the

[10]The strategies used in an alternative combinator could be also executed in parallel.

[11]`http://project-coco.uibk.ac.at/2022/`

[12]`http://project-coco.uibk.ac.at/2023/`

different categories. Detailed information can be obtained from competition full run web page which can be reached, for the different categories and years, from the following URL:

`http://cops.uibk.ac.at/results/`

Table 7. CONFident at the CoCo 2022 Full Run (left) and at the CoCo 2023 Full Run (right)

COPS cat.	Yes	No	Maybe	Solved	Total	Yes	No	Maybe	Solved	Total
TRS	108	138	331	246	577	109	138	330	247	577
CTRS	82	43	36	125	161	81	38	42	119	161
CS-TRS	25	87	64	112	176	49	83	44	132	176
CS-CTRS	88	41	158	129	287	94	47	146	141	287

CONFident obtained good results in the CTRS and CSR categories that we describe in the following.

Table 8. CTRS Category: CoCo 2022 Full Run (left) and CoCo 2023 Full Run (right)

Tool	Yes	No	Maybe	Solved	Total	Yes	No	Maybe	Solved	Total
ACP	54	26	81	80	161	54	25	82	79	161
CO3	57	32	72	89	161	57	35	69	92	161
CONFident	82	43	36	125	161	81	38	42	119	161

CONFident at the CTRS category of CoCo 2022 and 2023. Table 8 summarizes the results on the COPS collection of CTRSs used in the CoCo 2022 and 2023 *full-run* which we use below to provide an analysis of use of our processors. There was a bug in the 2022 version of the tool which was fixed in the 2023 version. From the second row in Table 8 we can see that a CTRS (actually COPS #1289, as can be seen on CoCo 2023 full-run web page) was proved confluent by CONFident 2022, but could *not* be handled by CONFident 2023. Also, 5 examples (actually, COPS #311, #312, #353, #524, and #1138) were proved *non-confluent* by CONFident 2022, but could *not* be handled by CONFident 2023. The reason is that the 2023 version of the tool could not deliver a proof within the $60''$ timeout of the competition. It can be checked that the five examples can be proved by the online version of CONFident if the default $120''$ timeout is used; furthermore, if a timeout of $60''$ is selected, the proofs are *not* obtained. By lack of time, we could not appropriately tune the 2023 version to reproduce on CoCo 2023 all good results obtained in CoCo 2022.

Taking CoCo 2023 as a reference, we see that CONFident solves almost 74% of the 161 CTRS problems in COPS mainly using the techniques described in [1, 22] orchestrated within the confluence framework described here. From the 161 examples, 28 of them (more than 17%) were proved by CONFident only. There were 5 problems (COPS #286, #311, #312, #353 and #406) that CONFident could not prove but were proved by ACP.

Still, CONFident obtained the first place in the CTRS category.

CONFident at the CSR category of CoCo 2022 and 2023. With respect to CSR, a *demonstration* subcategory of confluence of CSR was hosted as part of CoCo 2022 and a competitive subcategory was hosted at CoCo 2023, see `http://project-coco.uibk.ac.at/2022/categories/csr.php`. CONFident and ConfCSR participated in 2023. The results on the CoCo 2023 full-run for the CSR category, consisting of 176 CS-TRSs (COPS problems #1161 – #1164; #1167 – #1274; and #1298 – #1361) and 287 CS-CTRSs (COPS problems #1362 – #1648) are summarized in Table 9, see `http://cops.uibk` for complete details, where both CS-TRSs and CS-CTRSs are displayed in a single table. CONFident was buggy in the P_{JO} processor in 2022, but it was fixed and improved in the 2023 version.

Table 9. CSR Category: CoCo 2023 Full Run

Tool	Yes	No	Maybe	Erroneous	Solved	Total
ConfCSR	28	83	352	0	111	463
CONFident 2022	113	124	222	4	237	463
CONFident 2023	143	130	190	0	273	463

Table 10. Confluence of CSR: CS-TRSs (left) and CS-CTRSs (right)

Tool	Yes	No	Maybe	Solved	Total	Yes	No	Maybe	Solved	Total
ConfCSR	28	83	65	111	176	–	–	–	–	–
CONFident 2024	49	83	44	132	176	105	93	89	198	287

Unfortunately, due to a recently discovered bug in the implementation of the 2023 version of CONFident, processor $\mathsf{P}_{\mathcal{U}_{\text{conf}}}$ was used in proofs of confluence of CS-CTRSs, even though no theoretical result gives support to this yet. As a consequence, a number of CS-CTRSs were reported as confluent under no solid basis. We have fixed this problem and reproduced the full-run benchmarks with the new 2024 version of CONFident. Table 10 summarizes the obtained results. We separately show the results for CS-TRSs[13] and CS-CTRSs.[14] Although ConfCSR does not handle CS-CTRSs, for CS-TRSs we also include the results obtained by ConfCSR in the *full-run* of the CSR category of CoCo 2023.

Some observations follow:

1. The results on CS-TRSs obtained by CONFident 2023 & 2024, and summarized in Tables 9 and 10 are partly due to the implementation of results only recently reported in [52] which are included as part of processor P_{Orth} (not explicit in the description of the processor in Section 5.4.3, but included in the statistical account of Table 11).

2. Except for 8 cases, (COPS #1423, #1424, #1467, #1583, #1587, #1588, #1609 and #1610) all positive answers of CONFident 2023 for CS-CTRSs due to the buggy application of $\mathsf{P}_{\mathcal{U}_{\text{conf}}}$ have been confirmed by CONFident 2024 by using other techniques. This suggests that using

[13]Complete details in `http://zenon.dsic.upv.es/confident/benchmarks/fi24/cstrs/`

[14]Complete details in `http://zenon.dsic.upv.es/confident/benchmarks/fi24/csctrs/`

$\mathsf{P}_{\mathcal{U}_{\text{conf}}}$ with CS-CTRSs is probably correct; a formal proof of this conjecture should be provided, though.

3. After removing the use of $\mathsf{P}_{\mathcal{U}_{\text{conf}}}$ from CONFident 2024 proof strategy for CS-CTRSs, we are able to handle *more* CS-CTRSs: in 2023 we were able to (dis)prove confluence of $273 - 132 = 141$ CS-CTRSs. In contrast, CONFident 2024 solves 198 CS-CTRSs. Since no other change has been introduced, this shows the impact of the appropriate selection and order of use of processors in the proof strategy of automated analysis tools.

Table 11. Use of processors in the experiments

	#Y	#N	P_{EVar}	P_{Simp}	P_{Inl}	P_{MD}	$\mathsf{P}_{\mathcal{U}_{\text{conf}}}$	P_{Orth}	P_{WCR}	P_{SCR}	P_{KB}	P_{CanCR}	P_{HE}	P_{JO}
TRS	109	138	0	37	0	20	–	46	137	37	35	16	175	518
CTRS	81	38	0	51	24	8	34	72	44	0	16	0	39	71
CS-TRS	49	83	0	0	0	–	–	49	87	0	0	–	83	84
CS-CTRS	105	93	9	99	40	–	–	95	32	0	46	–	59	125
TOTAL	344	349	9	187	64	28	34	262	300	37	97	16	356	798

Use of processors. Table 11 summarizes the use of processors in the latest version of CONFident. We display the *number of uses* of each processor along the whole benchmark set. Some remarks are in order:

1. According to Tables 3,5, and 6:

   - *Proofs of confluence* eventually use the (sound) processors P_{Orth}, P_{HE}, or P_{KB} on CR-problems in the proof tree. This is because they are able to either (i) finish proof branches with a positive result (e.g., P_{Orth} and P_{KB}), or (ii) translate the original CR-problem into a WCR/SCR-problem (using P_{WCR} or P_{SCR}) which is then handled by P_{HE} and ultimately by P_{JO} to obtain a positive result.

   - Proofs of *non-confluence* are due to the use of (complete) processors on CR-problems like P_{EVar} (which finishes a proof branch with a negative result which is propagated to the root of the proof tree), P_{WCR} (which translates the CR-problem into a WCR-problem), and then P_{HE} (which returns JO-problems which can be qualified as *negative* by P_{JO}).

2. Regarding the second row (CTRSs): although CONFident does not implement any modularity result for CTRSs (yet), processor P_{MD} is used in some proofs with CTRSs, but only after transforming them into TRSs by applying other processors, e.g., P_{Simp}, P_{Inl}, or $\mathsf{P}_{\mathcal{U}_{\text{conf}}}$.

3. Processor P_{EVar} is used in some proofs with CS-CTRSs (fourth row) due to the presence of rules with extra variables in some of the COPS examples (see, e.g., `COPS/1367.trs`).

4. Some processors were not used in the competition (e.g., $\mathsf{P}_{\mathcal{U}}$ or P_{CanJ}, possibly due to their "low" position in the strategy; or P_{CnvJ}, which was not available yet). Thus, they are not mentioned in the table.

5. Some processors are used several times to solve a single example. For instance, in the CTRS category (see Table 8), the 38 proofs of non-confluence required 39 uses of P_{HE} due to one of the decompositions introduced by P_{MD}. Similarly, the 81 positive proofs obtained required $72+16 = 88$ uses of P_{Orth} or P_{KB} due to the remaining 7 decompositions introduced by P_{MD}.

Table 12. Use of heuristics and joinability methods within P_{JO}

	H1	H2	H3	J1	J2	J3	infChecker	Other
TRSs	0	4	0	10	211	189	58	47
CTRSs	2	0	2	5	21	0	40	6
CS-TRSs	0	4	0	5	78	0	0	0
CS-CTRSs	4	0	3	14	11	0	4	175

Use of heuristics and joinability techniques. Table 12 summarizes the use of heuristics (H1, H2, and H3 for Proposition 5.32, see Remark 5.33) and joinability techniques (J1, J2, and J3, see Remark 5.38) in the implementation of P_{JO} in CoCo 2022 (see Section 5.5). We display the *number of uses* of each heuristic or technique for each category. Besides, we report on uses of infChecker to prove joinability using other results in Proposition 5.35, or *other* techniques (e.g., use of Mace4 and Prover9). Some remarks are in order:

1. First row (TRSs): H2 is used with TRSs due to the use of P_{CanCR} which transforms a confluence problem for a TRS into a confluence problem for a CS-TRS (see the first row in Table 11). This may lead to compute LH_{μ}-critical pairs whose non-joinability is treated using Proposition Proposition 5.32 and hence the heuristics in Remark 5.33.

2. J3 (for strong joinability) is *not* used with CTRSs, CS-TRSs, or CS-CTRSs because, for SCR-problems, P_{HE} (which would eventually require it through a call to P_{JO}) is defined for linear TRSs only, see Section 5.3.

9. Related work

Several tools can be used to automatically prove and disprove confluence of TRSs and oriented CTRSs. ACP (Automated Confluence Prover) [53] implements a divide-and-conquer approach in two steps: first, a decomposition step is applied (based on modularity results); then, direct techniques are applied to each decomposition. ConCon [54] first tries to simplify rules and remove infeasible rules from the input system, then it employs a number of confluence criteria for (oriented) 3-CTRSs. ConCon uses several confluence criteria, and tree automata techniques on reachability to prove (in)feasibility of conditional parts. In parallel, ConCon tries to show non-confluence using conditional narrowing (and some other heuristics). CSI [55] uses a set of techniques (Knuth and Bendix' criterion, non-confluence criterion, order-sorted decomposition, development closed criterion, decreasing diagrams and extended rules) and a strategy language to combine them. CoLL-Saigawa [56] is the combination of two tools: CoLL and Saigawa. If the input system is left-linear, it uses CoLL; otherwise, it uses Saigawa. Among the techniques used by these tools are Hindley's commutation theorem together with the three commutation criteria, almost development closedness, rule labeling with weight function, Church-Rosser modulo AC, criteria based on different kinds of critical pairs, rule labeling, parallel closedness based on parallel critical pairs, simultaneous closedness, parallel-upside closedness, and outside closedness. CO3 [57] uses confluence (and termination) of $\mathcal{U}(\mathcal{R})$ and $\mathcal{U}_{opt}(\mathcal{R})$ (which is a variant of $\mathcal{U}(\mathcal{R})$, see [58, Section 7] for a discussion), in addition to narrowing trees for checking infeasibility of conditional parts in proofs of confluence of CTRSs. Hakusan [59] is a confluence tool for left-linear TRSs that analyzes confluence by using two compositional confluence criteria [60]. It returns certified outputs for rule labeling and develops a novel reduction method.

To demonstrate confluence of CSR, apart from CONFident, only ConfCSR [61] is able to prove confluence of CS-TRSs by using, essentially, the results in [25] while relying on AProVE to prove termination of CSR.

Although the previous tools also use combinations of different techniques to obtain proofs of confluence, to the best of our knowledge, none of them formalize such a combination as done in this paper, where the confluence framework provides a systematic way to organize and decompose proofs of (local, strong) confluence problems by transforming them into other (possibly simpler) problems. Also, an important difference between CONFident and all previous tools is the encoding of (non-)joinability of (conditional) pairs as combinations of *(in)feasibility* problems [16]. As explained in [62] and further developed in [1, 22, 25], this is possible due to the logic-based treatment of rewrite systems as first-order theories. This is also the key for CONFident to be able to smoothly handle join, oriented, and semi-equational CTRSs, something which is also a novel feature of the tool in comparison with the aforementioned tools. Furthermore, as far as we know, CONFident is the only tool implementing confluence criteria depending on *termination* of CTRSs rather than more restrictive termination properties like quasi-decreasingness [7, Def. 7.2.39], see [1, Section 8] for a deeper discussion and further motivation.

10. Conclusions and future work

CONFident is a tool which is able to automatically prove and disprove confluence of variants of rewrite systems: TRSs, CS-TRSs, and Join, Oriented, and Semi-Equational CTRSs and CS-CTRSs. The proofs are obtained by combining different techniques in what we call *Confluence Framework*, which we have introduced here, where different variants of *confluence* and *joinability* problems are handled (simplified, transformed, etc.) by means of *processors*, which can be freely combined to obtain the proofs which are displayed as a *proof tree* (Definition 4.5 and Theorem 4.6). In this paper, we have introduced 16 processors which can be used in the confluence framework. Some of them just integrate existing results by other authors as processors to be used in the confluence framework (see the description of the processors and the corresponding references to the used methods and supporting results in Section 5). We believe that other results not considered here, possibly involving

- other approaches like the *compositional approach* developed in [60] which reformulates and somehow *unifies* a number of well-known confluence results for TRSs from an abstract compositional presentation, and then obtains improvements from them by applying the technique developed by the authors; or

- other kind of confluence problems (e.g., *ground* confluence, i.e., confluence of rewriting restricted to *ground* terms only); or
- rewriting forms (e.g., *rewriting modulo* a set of equations [63], *nominal rewriting* [64], etc.), see, e.g., [65, 66, 67, 68, 69, 70], etc.,

could also be ported into the confluence framework. This may involve the definition of new problems: for instance $GCR(\mathcal{R})$ for ground confluence of a GTRS $\mathcal{R}$ and $ECR(\mathcal{R})$ for confluence modulo of an *Equational* GTRS $\mathcal{R}$ (which, essentially, are GTRSs extended with a set of conditional equations, see [70, Definition 10]). Whether new kind of problems have been considered or not, we think that available or forthcoming techniques for proving such problems positive or negative can be integrated in the confluence framework by means of appropriate processors so that they can be smoothly combined with other processors implementing different techniques to obtain a more powerful framework to prove confluence.

We have shown how to implement the confluence framework using the declarative programming language Haskell. CONFident has proved to be a powerful tool for proving confluence of CTRSs. This is witnessed by the first position obtained in the CTRS category of CoCo 2021, 2022 and 2023 and also by the good results obtained in the CSR category at CoCo 2023, where both CS-TRSs and CS-CTRSs were considered.

Future work. First of all, CONFident should be extended to deal with arbitrary GTRSs. Providing direct access to proofs of *local* and *strong* confluence (and even joinability of terms) would also be interesting by using the specific problems introduced here and their treatment within the confluence framework. Also, some features of the discussed processors are not implemented yet (e.g., modularity of local and strong confluence of TRSs) and we should also include existing results for CTRSs (see, e.g., [33, Table 8.4]). Besides, as far as we know, modularity of confluence of CSR, either for CS-TRSs or for CS-CTRSs has not been investigated yet. Also considering confluence of order-sorted rewrite systems [71] and equational rewrite systems [66] could be an interesting task for future work.

Acknowledgements. We thank the referees for their detailed comments and suggestions that led to many improvements in the paper. We thank Vincent van Oostrom for providing early access to [38] and also for his comments on [18]. Finally, we thank the organizers of the Confluence Competition and maintainers of the COPS database for their (successful) effort to promote the research on confluence and automation of program analysis.

Raúl Gutiérrez was partially supported by the grant PID2021-122830OB-C44 funded by CIN/AEI/10.13039/501100011033 and by the project ”ERDF - a way to build Europe”.

Salvador Lucas and Miguel Vítores were partially supported by grant PID2021-122830OB-C42 funded by MCIN/AEI/10.13039/501100011033 and by “ERDF - a way of making Europe”. Salvador Lucas was also supported by the grant CIPROM/2022/6 funded by Generalitat Valenciana.

References

[1] Lucas S. Local confluence of conditional and generalized term rewriting systems. *Journal of Logical and Algebraic Methods in Programming*, 2024. **136**:paper 100926, pages 1–23. doi:10.1016/j.jlamp.2023.100926.

[2] Lucas S. Context-sensitive Rewriting. *ACM Comput. Surv.*, 2020. **53**(4):78:1–78:36. doi:10.1145/3397677.

[3] Baader F, Nipkow T. Term Rewriting and All That. Cambridge University Press, 1998. ISBN 978-0-521-45520-6. doi:10.1017/CBO9781139172752.

[4] Kaplan S. Conditional Rewrite Rules. *Theor. Comput. Sci.*, 1984. **33**:175–193. doi:10.1016/0304-3975(84)90087-2.

[5] Lucas S. Applications and extensions of context-sensitive rewriting. *Journal of Logical and Algebraic Methods in Programming*, 2021. **121**:100680. doi:10.1016/j.jlamp.2021.100680.

[6] Gmeiner KS. Transformational Approaches for Conditional Term Rewrite Systems. Ph.D. thesis, Faculty of Informatics, Vienna University of Technology, 2014.

[7] Ohlebusch E. Advanced Topics in Term Rewriting. Springer, 2002. ISBN 978-0-387-95250-5. doi:10.1007/978-1-4757-3661-8.

[8] Giesl J, Thiemann R, Schneider-Kamp P. The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs. In: Baader F, Voronkov A (eds.), Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Proceedings, volume 3452 of *Lecture Notes in Computer Science*. Springer, 2004 pp. 301–331. doi:10.1007/978-3-540-32275-7_21.

[9] Giesl J, Thiemann R, Schneider-Kamp P, Falke S. Mechanizing and Improving Dependency Pairs. *J. Autom. Reasoning*, 2006. **37**(3):155–203. doi:10.1007/s10817-006-9057-7.

[10] Blanqui F, Genestier G, Hermant O. Dependency Pairs Termination in Dependent Type Theory Modulo Rewriting. In: Geuvers H (ed.), 4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany, volume 131 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019 pp. 9:1–9:21. doi:10.4230/LIPICS.FSCD.2019.9.

[11] Falke S, Kapur D. Dependency Pairs for Rewriting with Non-free Constructors. In: Pfenning F (ed.), Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings, volume 4603 of *Lecture Notes in Computer Science*. Springer, 2007 pp. 426–442. doi:10.1007/978-3-540-73595-3_32. URL `https://doi.org/10.1007/978-3-540-73595-3_32`.

[12] Falke S, Kapur D. Dependency Pairs for Rewriting with Built-In Numbers and Semantic Data Structures. In: Voronkov A (ed.), Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings, volume 5117 of *Lecture Notes in Computer Science*. Springer, 2008 pp. 94–109. doi:10.1007/978-3-540-70590-1_7.

[13] Gutiérrez R, Lucas S. Proving Termination in the Context-Sensitive Dependency Pair Framework. In: Ölveczky PC (ed.), Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Held as a Satellite Event of ETAPS 2010, Revised Selected Papers, volume 6381 of *Lecture Notes in Computer Science*. Springer, 2010 pp. 18–34. doi:10.1007/978-3-642-16310-4_3.

[14] Lucas S, Meseguer J, Gutiérrez R. The 2D Dependency Pair Framework for conditional rewrite systems. Part I: Definition and basic processors. *J. Comput. Syst. Sci.*, 2018. **96**:74–106. doi:10.1016/j.jcss.2018.04.002.

[15] Lucas S. Termination of Generalized Term Rewriting Systems. In: Rehof J (ed.), 9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024), volume 299 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2024 pp. 29:1–29:18. doi:10.4230/LIPIcs.FSCD.2024.29.

[16] Gutiérrez R, Lucas S. Automatically Proving and Disproving Feasibility Conditions. In: Peltier N, Sofronie-Stokkermans V (eds.), Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Proceedings, Part II, volume 12167 of *Lecture Notes in Computer Science*. Springer, 2020 pp. 416–435. doi:10.1007/978-3-030-51054-1_27.

[17] Kaplan S. Fair Conditional Term Rewriting Systems: Unification, Termination, and Confluence. In: Kreowski H (ed.), Recent Trends in Data Type Specification, 3rd Workshop on Theory and Applications of Abstract Data Types, 1984, Selected Papers, volume 116 of *Informatik-Fachberichte*. Springer, 1984 pp. 136–155. doi:10.1007/978-3-662-09691-8_11.

[18] Gutiérrez R, Vítores M, Lucas S. Confluence Framework: Proving Confluence with CONFident. In: Villanueva A (ed.), Logic-Based Program Synthesis and Transformation - 32nd International Symposium, LOPSTR 2022, Proceedings, volume 13474 of *Lecture Notes in Computer Science*. Springer, 2022 pp. 24–43. doi:10.1007/978-3-031-16767-6_2.

[19] Terese. Term rewriting systems, volume 55 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 2003. ISBN 978-0-521-39115-3.

[20] Fitting M. First-Order Logic and Automated Theorem Proving, Second Edition. Graduate Texts in Computer Science. Springer, 1996. ISBN 978-1-4612-7515-2. doi:10.1007/978-1-4612-2360-3.

[21] Mendelson E. Introduction to mathematical logic (4. ed.). Chapman and Hall, 1997. ISBN 978-0-412-080830-7.

[22] Gutiérrez R, Lucas S, Vítores M. Confluence of Conditional Rewriting in Logic Form. In: Bojańczy M, Chekuri C (eds.), 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021), volume 213 of *Leibniz International Proceedings in Informatics (LIPIcs)*. ISBN 978-3-95977-215-0, 2021 pp. 44:1–44:18. doi:10.4230/LIPIcs.FSTTCS.2021.44.

[23] Avenhaus J, Loría-Sáenz C. On Conditional Rewrite Systems with Extra Variables and Deterministic Logic Programs. In: Pfenning F (ed.), Logic Programming and Automated Reasoning, 5th International Conference, LPAR'94, Proceedings, volume 822 of *Lecture Notes in Computer Science*. Springer, 1994 pp. 215–229. doi:10.1007/3-540-58216-9_40.

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

[25] Lucas S, Vítores M, Gutiérrez R. Proving and disproving confluence of context-sensitive rewriting. *Journal of Logical and Algebraic Methods in Programming*, 2022. **126**:100749. doi:10.1016/j.jlamp.2022.100749.

[26] Sternagel T. Reliable Confluence Analysis of Conditional Term Rewrite Systems. Ph.D. thesis, Faculty of Mathematics, Computer Science and Physics, University of Innsbruck, 2017.

[27] Gramlich B. Modularity in term rewriting revisited. *Theor. Comput. Sci.*, 2012. **464**:3–19. doi:10.1016/J.TCS.2012.09.008.

[28] Middeldorp A. Modular Properties of Conditional Term Rewriting Systems. *Inf. Comput.*, 1993. **104**(1):110–158. doi:10.1006/inco.1993.1027.

[29] Toyama Y. On the Church-Rosser property for the direct sum of term rewriting systems. *J. ACM*, 1987. **34**(1):128–143. doi:10.1145/7531.7534.

[30] Kurihara M, Ohuchi A. Modularity of Simple Termination of Term Rewriting Systems with Shared Constructors. *Theor. Comput. Sci.*, 1992. **103**(2):273–282. doi:10.1016/0304-3975(92)90015-8.

[31] Middeldorp A, Toyama Y. Completeness of Combinations of Constructor Systems. *J. Symb. Comput.*, 1993. **15**(3):331–348. doi:10.1006/JSCO.1993.1024.

[32] Ohlebusch E. Modular Properties of Composable Term Rewriting Systems. *J. Symb. Comput.*, 1995. **20**(1):1–41. doi:10.1006/JSCO.1995.1036.

[33] Ohlebusch E. On the Modularity of Confluence of Constructor-Sharing Term Rewriting Systems. In: Tison S (ed.), Trees in Algebra and Programming - CAAP'94, 19th International Colloquium, Edinburgh, UK, April 11-13, 1994, Proceedings, volume 787 of *Lecture Notes in Computer Science*. Springer, 1994 pp. 261–275. doi:10.1007/BFb0017487.

[34] Ohlebusch E. Modular Properties Of Composable Term Rewriting Systems. Ph.D. thesis, FUniversität Bielefeld, 1994.

[35] Raoult J, Vuillemin J. Operational and Semantic Equivalence Between Recursive Programs. *J. ACM*, 1980. **27**(4):772–796. doi:10.1145/322217.322229.

[36] Middeldorp A. Modular Properties Of Term Rewriting Systems. Ph.D. thesis, Vrije Universiteit te Amsterdam, 1990.

[37] Huet GP. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *J. ACM*, 1980. **27**(4):797–821. doi:10.1145/322217.322230.

[38] van Oostrom V. The Z-property for left-linear term rewriting via convective context-sensitive completeness. In: 12th International Workshop on Confluence, IWC 2023. 2023 pp. 38–43.

[39] Marchiori M. Unravelings and Ultra-properties. In: Hanus M, Rodríguez-Artalejo M (eds.), Algebraic and Logic Programming, 5th International Conference, ALP'96, Aachen, Germany, September 25-27, 1996, Proceedings, volume 1139 of *Lecture Notes in Computer Science*. Springer, 1996 pp. 107–121. doi:10.1007/3-540-61735-3_7.

[40] Gmeiner K, Gramlich B, Schernhammer F. On Soundness Conditions for Unraveling Deterministic Conditional Rewrite Systems. In: Tiwari A (ed.), 23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, volume 15 of *Leibniz International Proceedings in Informatics (LIPIcs)*. 2012 pp. 193–208. doi:10.4230/LIPIcs.RTA.2012.193.

[41] Nishida N, Sakai M, Sakabe T. Soundness of Unravelings for Conditional Term Rewriting Systems via Ultra-Properties Related to Linearity. *Log. Methods Comput. Sci.*, 2012. **8**(3). doi:10.2168/LMCS-8(3:4)2012.

[42] Gmeiner K, Nishida N, Gramlich B. Proving Confluence of Conditional Term Rewriting Systems via Unravelings. In: 2nd International Workshop on Confluence, IWC 2013. 2013 pp. 35–39.

[43] Suzuki T, Middeldorp A, Ida T. Level-Confluence of Conditional Rewrite Systems with Extra Variables in Right-Hand Sides. In: Hsiang J (ed.), Rewriting Techniques and Applications, 6th International Conference, RTA-95, Proceedings, volume 914 of *Lecture Notes in Computer Science*. Springer, 1995 pp. 179–193. doi:10.1007/3-540-59200-8_56.

[44] Gramlich B, Lucas S. Generalizing Newman's Lemma for Left-Linear Rewrite Systems. In: Pfenning F (ed.), Term Rewriting and Applications, 17th International Conference, RTA 2006, Proceedings, volume 4098 of *Lecture Notes in Computer Science*. Springer, 2006 pp. 66–80. doi:10.1007/11805618_6.

[45] Knuth DE, Bendix PE. Simple Word Problems in Universal Algebra. In: Leech J (ed.), Computational Problems in Abstract Algebra. Pergamon Press, 1970 pp. 263–297.

[46] Dershowitz N, Okada M, Sivakumar G. Confluence of Conditional Rewrite Systems. In: Kaplan S, Jouannaud J (eds.), Conditional Term Rewriting Systems, 1st International Workshop, Proceedings, volume 308 of *Lecture Notes in Computer Science*. Springer, 1987 pp. 31–44. doi:10.1007/3-540-19242-5_3.

[47] Bergstra JA, Klop JW. Conditional Rewrite Rules: Confluence and Termination. *J. Comput. Syst. Sci.*, 1986. **32**(3):323–362. doi:10.1016/0022-0000(86)90033-4.

[48] Gutiérrez R, Lucas S. Automatic Generation of Logical Models with AGES. In: Fontaine P (ed.), Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Proceedings, volume 11716 of *Lecture Notes in Computer Science*. Springer, 2019 pp. 287–299. doi: 10.1007/978-3-030-29436-6_17.

[49] McCune W. Prover9 & Mace4. Technical report, University of New Mexico, 2005–2010. URL `http://www.cs.unm.edu/~mccune/prover9/`.

[50] Rapp F, Middeldorp A. FORT 2.0. In: Galmiche D, Schulz S, Sebastiani R (eds.), Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Proceedings, volume 10900 of *Lecture Notes in Computer Science*. Springer, 2018 pp. 81–88. doi: 10.1007/978-3-319-94205-6_6.

[51] Swierstra W. Data types à la carte. *J. Funct. Program.*, 2008. **18**(4):423–436. doi:10.1017/S0956796808006758.

[52] Lucas S. Orthogonality of Generalized Term Rewriting Systems Submitted to the 13th International Workshop on Confluence, IWC 2024.

[53] Aoto T, Yoshida J, Toyama Y. Proving Confluence of Term Rewriting Systems Automatically. In: Treinen R (ed.), Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Proceedings, volume 5595 of *Lecture Notes in Computer Science*. Springer, 2009 pp. 93–102. doi: 10.1007/978-3-642-02348-4_7.

[54] Sternagel C. CoCo 2020 Participant: ConCon 1.10. In: Ayala-Rincón M, Mirmram S (eds.), Proc. of the 9th International Workshop on Confluence, IWC'20. 2020 p. 65.

[55] Nagele J, Felgenhauer B, Middeldorp A. CSI: New Evidence - A Progress Report. In: de Moura L (ed.), Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Proceedings, volume 10395 of *Lecture Notes in Computer Science*. Springer, 2017 pp. 385–397. doi: 10.1007/978-3-319-63046-5_24.

[56] Shintani K, Hirokawa N. CoLL-Saigawa 1.6: A Joint Confluence Tool. In: Mirmram S, Rocha C (eds.), Proc. of the 10th International Workshop on Confluence, IWC'21. 2021 p. 51.

[57] Nishida N, Kuroda T, Yanagisawa M, Gmeiner K. CO3: a COnverter for proving COnfluence of COnditional TRSs. In: 4th International Workshop on Confluence, IWC 2015. 2015 p. 42.

[58] Schernhammer F, Gramlich B. Characterizing and proving operational termination of deterministic conditional term rewriting systems. *J. Log. Algebraic Methods Program.*, 2010. **79**(7):659–688. doi: 10.1016/j.jlap.2009.08.001.

[59] Shintani K, Hirokawa N. Hakusan 0.8: A Confluence Tool. In: 12th International Workshop on Confluence, IWC 2023. 2023 p. 65.

[60] Shintani K, Hirokawa N. Compositional Confluence Criteria. In: Felty AP (ed.), 7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel, volume 228 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022 pp. 28:1–28:19. doi:10.4230/LIPICS.FSCD.2022.28.

[61] Stevanovic F, Mitterwallner F. CoCo 2023 Participant: ConfCSR. In: 12th International Workshop on Confluence, IWC 2023. 2023 p. 63.

[62] Lucas S. Proving semantic properties as first-order satisfiability. *Artif. Intell.*, 2019. **277**. doi:10.1016/j.artint.2019.103174.

[63] Peterson GE, Stickel ME. Complete Sets of Reductions for Some Equational Theories. *J. ACM*, 1981. **28**(2):233–264. doi:10.1145/322248.322251.

[64] Fernández M, Gabbay M. Nominal rewriting. *Inf. Comput.*, 2007. **205**(6):917–965. doi:10.1016/J.IC.2006.12.002. URL `https://doi.org/10.1016/j.ic.2006.12.002`.

[65] Durán F, Meseguer J. On the Church-Rosser and coherence properties of conditional order-sorted rewrite theories. *J. Log. Algebraic Methods Program.*, 2012. **81**(7-8):816–850. doi:10.1016/j.jlap.2011.12.004.

[66] Jouannaud J. Confluent and Coherent Equational Term Rewriting Systems: Application to Proofs in Abstract Data Types. In: Ausiello G, Protasi M (eds.), CAAP'83, Trees in Algebra and Programming, 8th Colloquium, Proceedings, volume 159 of *Lecture Notes in Computer Science*. Springer, 1983 pp. 269–283. doi:10.1007/3-540-12727-5_16.

[67] Jouannaud J, Kirchner H. Completion of a Set of Rules Modulo a Set of Equations. *SIAM J. Comput.*, 1986. **15**(4):1155–1194. doi:10.1137/0215084.

[68] Kikuchi K. Ground Confluence and Strong Commutation Modulo Alpha-Equivalence in Nominal Rewriting. In: Seidl H, Liu Z, Pasareanu CS (eds.), Theoretical Aspects of Computing - ICTAC 2022 - 19th International Colloquium, Tbilisi, Georgia, September 27-29, 2022, Proceedings, volume 13572 of *Lecture Notes in Computer Science*. Springer, 2022 pp. 255–271. doi:10.1007/978-3-031-17715-6_17.

[69] Kikuchi K, Aoto T. Confluence and Commutation for Nominal Rewriting Systems with Atom-Variables. In: Fernández M (ed.), Logic-Based Program Synthesis and Transformation - 30th International Symposium, LOPSTR 2020, Bologna, Italy, September 7-9, 2020, Proceedings, volume 12561 of *Lecture Notes in Computer Science*. Springer, 2020 pp. 56–73. doi:10.1007/978-3-030-68446-4_3.

[70] Lucas S. Confluence of Conditional Rewriting Modulo. In: Murano A, Silva A (eds.), 32nd EACSL Annual Conference on Computer Science Logic (CSL 2024), volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany. ISBN 978-3-95977-310-2, 2024 pp. 37:1–37:21. doi:10.4230/LIPIcs.CSL.2024.37.

[71] Goguen JA, Meseguer J. Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. *Theor. Comput. Sci.*, 1992. **105**(2):217–273. doi:10.1016/0304-3975(92)90302-V.

A. Proof of Proposition 5.3

Proposition 5.3 *Let $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ be a GTRS, $\alpha \in R$, i, and x as in Definition 5.2. Let s and t be terms.*

1. *If $s \to_{\mathcal{R}} t$, then $s \to^*_{\mathcal{R}_{\alpha,x,i}} t$.*
2. *If $s \to_{\mathcal{R}_{\alpha,x,i}} t$, then $s \to_{\mathcal{R}} t$.*

Proof:
As for the *first claim* ($s \to_{\mathcal{R}} t$ implies $s \to^*_{\mathcal{R}_{\alpha,x,i}} t$), by using the version for GTRSs of [1, Theorem 10] for CTRSs (see [1, Section 7.5]), $s \to_{\mathcal{R}} t$ iff $\overline{\mathcal{R}} \vdash s^{\downarrow} \to t^{\downarrow}$. We proceed by induction on the length $n \geq 0$ of a *Hilbert-like* proof of $s^{\downarrow} \to t^{\downarrow}$ from $\overline{\mathcal{R}}$. Base: $n = 0$. With $s^{\downarrow} \to t^{\downarrow}$ we are using (HC)$_\alpha$ for some unconditional rule $\alpha : \ell \to r \in R$ such that $s^{\downarrow} = \sigma(\ell)$ and $t^{\downarrow} = \sigma(r)$. Since all unconditional rules in $\mathcal{R}$ are also in $\mathcal{R}_{\alpha,x,i}$, by also using (Rf) and (Co) we conclude $s^{\downarrow} \to^*_{\mathcal{R}_{\alpha,x,i}} t^{\downarrow}$, as required. Induction step: $n > 0$. We have two possibilities:

1. A sentence (Pr)$_{f,i}$ is used for some $f \in \mathcal{F}$ and $i \in \mu(f)$. Then, $s^{\downarrow} = f(s_1, \ldots, s_i, \ldots, s_k)$ and $t^{\downarrow} = f(s_1, \ldots, t_i, \ldots, s_k)$ for some (ground) terms $s_1, \ldots, s_k$ and t_i and $\overline{\mathcal{R}} \vdash s_i \to t_i$ has been proved in less than n steps. By the induction hypothesis, $\overline{\mathcal{R}_{\alpha,x,i}} \vdash s_i \to t_i$ holds as well. Thus, by using again (Pr)$_{f,i}$ (which is part of $\overline{\mathcal{R}_{\alpha,x,i}}$), we conclude $s^{\downarrow} \to_{\mathcal{R}_{\alpha,x,i}} t^{\downarrow}$.

2. A sentence (HC)$_\beta$ has been used for some $\beta \in R$. If $\beta \neq \alpha$, then β is also a rule of $\mathcal{R}_{\alpha,x,i}$ and by using the induction hypothesis, we conclude $s^{\downarrow} \to_{\mathcal{R}_{\alpha,x,i}} t^{\downarrow}$. Otherwise, $\beta = \alpha$ and $s^{\downarrow} = \varsigma(\ell)$ and $t^{\downarrow} = \varsigma(r)$ for some substitution ς, and also (by the induction hypothesis and because α is an O-rule), for all $1 \leq j \leq n$,

$$\varsigma(s_j) \to^*_{\mathcal{R}_{\alpha,x,i}} \varsigma(t_j) \tag{50}$$

In $\mathcal{R}_{\alpha,x,i}$ for $\sigma = \{x \mapsto s_i\}$, we have $\alpha_{x,i} : \ell \to \sigma(r) \Leftarrow \sigma(s_1) \approx t_1, \cdots, \sigma(s_{i-1}) \approx t_{i-1}, \sigma(s_{i+1}) \approx t_{i+1}, \cdots, \sigma(s_n) \approx t_n$ instead of α. Note that $\alpha_{x,i}$ contains $n-1$ conditions. For all $1 \leq j \leq n$, (a) if $x \notin \mathcal{V}ar(s_j)$, then $\sigma(s_j) = s_j$ and we have $\varsigma(s_j) = \varsigma(\sigma(s_j)) \to^* \varsigma(t_j)$; and (b) if $x \in \mathcal{V}ar(s_j)$, then we can write $s_j = C_j[x]_{P_j}$ for some context C_j and set P_j of positions of x in s_j and therefore, $\sigma(s_j) = C_j[s_i]_{P_j}$. Since, by (50), we have $\varsigma(s_i) \to^*_{\mathcal{R}_{\alpha,x,i}} \varsigma(t_i) = \varsigma(x)$ and also (by (13)) x is not frozen in s_j (and it does not occur in s_i), we have

$$\begin{array}{rcl}
\varsigma(\sigma(s_j)) & = & \varsigma(C_j[s_i]_{P_j}) \\
& = & \varsigma(C_j)[\varsigma(s_i)]_{P_j} \\
& \to^*_{\mathcal{R}_{\alpha,x,i}} & \varsigma(C_j)[\varsigma(x)]_{P_j} \\
& = & \varsigma(C_j[x]_{P_j}) \\
& = & \varsigma(s_j) \\
& \to^*_{\mathcal{R}_{\alpha,x,i}} & \varsigma(t_j)
\end{array}$$

where the last rewriting sequence is proved by using (50) again, now for $\varsigma(s_j)$. Therefore, all conditions $\sigma(s_j) \to t_j$, $1 \leq j \leq n$, $j \neq i$ in α, x, i are satisfied by ς. Thus, since $x \notin \mathcal{V}ar(\ell)$, we have $s^{\downarrow} = \varsigma(\ell) \to_{\mathcal{R}_{\alpha,x,i}} \varsigma(r)$. Again, (a) if $x \notin \mathcal{V}ar(r)$, then $\sigma(r) = r$ and we have $\varsigma(r) = t^{\downarrow}$, as required. Otherwise, if (b) $x \in \mathcal{V}ar(r)$, then $r = C[x]_P$ for some context C and set P of positions of x in r, and $\sigma(r) = C[s_i]_P$. Since (by (13)) x is *not* frozen in r (i.e., $P \subseteq \mathcal{P}os^{\mu}(r)$), we have

$$\varsigma(\sigma(r)) = \varsigma(C[s_i]_P) = \varsigma(C)[\varsigma(s_i)]_P \to^*_{\mathcal{R}_{\alpha,x,i}} \varsigma(C)[\varsigma(x)]_P = \varsigma(C[x]_P) = \varsigma(r) = t^{\downarrow}$$

Therefore, $s^{\downarrow} \to^*_{\mathcal{R}_{\alpha,x,i}} t^{\downarrow}$, as required.

Regarding the *second claim* ($s \to_{\mathcal{R}_{\alpha,x,i}} t$ implies $s \to_{\mathcal{R}} t$), we proceed by induction on length $n \geq 0$ of a *Hilbert-like* proof of $s^{\downarrow} \to t^{\downarrow}$ from $\overline{\mathcal{R}_{\alpha,x,i}}$. Base: $n = 0$. With $s^{\downarrow} \to t^{\downarrow}$ we are using $(\text{HC})_\alpha$ for some unconditional rule $\alpha : \lambda \to \rho$ such that $s^{\downarrow} = \varsigma(\lambda)$ and $t^{\downarrow} = \varsigma(\rho)$. The only possibility for an unconditional rule in $\mathcal{R}_{\alpha,x,i}$ of *not* being in $\mathcal{R}$ is $\alpha_{x,i}$, provided that the conditional part of α consists of a single condition $s \approx x$. In this case, $\alpha_{x,i}$ is $\ell \to r[s]_P$ where P are the positions of x in r, i.e., $r = r[x]_P$. Hence, $s^{\downarrow} = \varsigma(\ell)$ and $t^{\downarrow} = \varsigma(r[s]_P)$. Since $x \notin \mathcal{V}ar(\ell, s)$, we can assume that ς does not instantiate x. Define a substitution ς' as follows: $\varsigma'(x) = \varsigma(s)$ and for all $y \in \mathcal{X} - \{x\}$, $\varsigma'(y) = \varsigma(y)$. Note that $\varsigma'(\ell) = \varsigma(\ell)$ and $\varsigma'(s) = \varsigma(s)$. Then, the only condition $s \approx x$ in the conditional part of α is trivially satisfied by ς'. Therefore, since x is not frozen in r (by (13)), we have

$$s^{\downarrow} = \varsigma(\ell) = \varsigma'(\ell) \to_\alpha \varsigma'(r) = \varsigma'(r[x]_P) = \varsigma'(r)[\varsigma(s)]_P = \varsigma(r)[\varsigma(s)]_P = \varsigma(r[s]_P) = t^{\downarrow}$$

as desired, as the conditional part $s \approx x$ of α is satisfied by ς': $\varsigma'(s) = \varsigma(s) = \varsigma'(x)$. Induction step: $n > 0$. We have two possibilities:

1. A sentence $(\text{Pr})_{f,i}$ is used for some $f \in \mathcal{F}$ and $i \in \mu(f)$. Analogous to the corresponding case of the first claim.

2. A sentence $(\text{HC})_\beta$ has been used for some conditional rule β. If $\beta \neq \alpha_{x,i}$, then $\beta \in R$; hence $s^{\downarrow} \to_{\mathcal{R}} t^{\downarrow}$. Otherwise, if $\beta = \alpha_{x,i}$, then $s^{\downarrow} = \varsigma(\ell) \to_{\mathcal{R}_{\alpha,x,i}} \varsigma(r[s]_P) = t^{\downarrow}$ for some substitution ς such that $\varsigma(s_j[s]_{P_j}) \to^*_{\mathcal{R}_{\alpha,x,i}} \varsigma(t_j)$ holds for all $1 \leq j \leq n$, $j \neq i$. By the induction hypothesis (and repeatedly using (Co) and (Rf)), $\varsigma(s_j[s]_{P_j}) \to^*_{\mathcal{R}} \varsigma(t_j)$ holds for all $1 \leq j \leq n$, $j \neq i$. Define ς' as follows: $\varsigma'(y) = \varsigma(y)$ if $y \neq x$, and $\varsigma'(x) = \varsigma(s)$. For all $1 \leq j \leq n$, $j \neq i$,

$$\varsigma'(s_j) = \varsigma'(s_j[x]_{P_j}) = \varsigma(s_j)[\varsigma(x)]_{P_j} = \varsigma(s_j)[s]_{P_j} = \varsigma(s_j[s]_{P_j}) \to^*_{\mathcal{R}} \varsigma(t_j) = \varsigma'(t_j)$$

because $x \notin \mathcal{V}ar(s)$ and $x \notin \mathcal{V}ar(t_j)$. We also have

$$\varsigma'(s_i) = \varsigma'(s) = \varsigma(s) = \varsigma'(x)$$

Therefore, we finally have

$$s^{\downarrow} = \varsigma'(\ell) \to_{\mathcal{R}} \varsigma'(r) = \varsigma'(r[x]_P) = \varsigma(r)[\varsigma(s)]_p = \varsigma(r[s]_P) = t^{\downarrow}$$

as required. □