**Journal of Automated Reasoning manuscript No.**
(will be inserted by the editor)

# Certified Confluence of Conditional Rewriting

**Christian Sternagel (Corresponding author) ·**
**Thomas Sternagel**

Received: date / Accepted: date

**Abstract** Conditional term rewriting is a Turing-complete model of computation that allows for more direct translations between programs and rewrite systems than is typically the case for plain term rewriting without conditions. On the one hand, this alleviates program verification. On the other hand, properties of the corresponding conditional term rewrite systems, like confluence, are typically harder to establish than for plain term rewrite systems.

However, there are several automated tools that try to establish confluence of conditional term rewrite systems. In order to make the results of such tools more reliable, we formalize most of the state-of-the-art techniques they use for proving confluence of conditional term rewrite systems using the proof assistant Isabelle/HOL. Using these results, we provide a fully verified certifier that allows us to validate proofs that are generated by automated tools. Moreover, we evaluate our approach on standard benchmarks.

**Keywords** Formalization · Certification · Automation · Confluence · Conditional Term Rewriting

## 1 Introduction

Term rewriting is a well-studied and Turing-complete model of computation used in many different areas of computer science. One such area is program verification, where we typically first translate a given computer program into a corresponding term rewrite system (TRS for short) and then apply term rewriting techniques in order to establish properties like confluence and termination. Of course this approach only allows us to transfer an inferred property from the TRS to the original program, provided the employed translation is sound with regard to the property.

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

C. Sternagel, T. Sternagel
University of Innsbruck
E-mail: {christian, thomas}.sternagel@uibk.ac.at

```
qsort []       =  []
qsort (x:xs)   =  qsort us ++ [x] ++ qsort vs
  where
    (us, vs) = split x xs

    split x []                  =  ([], [])
    split x (y:ys) | y <= x     =  (y:us, vs)
                   | otherwise  =  (us, y:vs)
      where
        (us, vs) = split x ys
```

Listing 1: Haskell implementation of quicksort

The more direct the translation, the easier it is to guarantee its soundness. Typically, conditional term rewrite systems (CTRSs for short) allow for more direct translations than plain TRSs.

Consider, for example, the Haskell program from Listing 1 that implements a version of the quicksort algorithm for lists.

It can be translated into the following CTRS, where `where`-clauses as well as pattern guards (like "`| y <= x`") of the Haskell program are directly captured by conditions:

$$\begin{aligned}
\mathsf{qsort}(\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{qsort}(x : xs) &\to \mathsf{qsort}(us) \mathbin{++} (x : \mathsf{nil}) \mathbin{++} \mathsf{qsort}(vs)\\
&\Leftarrow \mathsf{split}(x, xs) \twoheadrightarrow \mathsf{pair}(us, vs)
\end{aligned}$$

$$\begin{aligned}
\mathsf{split}(x, \mathsf{nil}) &\to \mathsf{pair}(\mathsf{nil}, \mathsf{nil})\\
\mathsf{split}(x, y : ys) &\to \mathsf{pair}(y : us, vs)\\
&\Leftarrow \mathsf{split}(x, ys) \twoheadrightarrow \mathsf{pair}(us, vs),\ \mathsf{leq}(y, x) \twoheadrightarrow \mathsf{true}\\
\mathsf{split}(x, y : ys) &\to \mathsf{pair}(ys, y : vs)\\
&\Leftarrow \mathsf{split}(x, ys) \twoheadrightarrow \mathsf{pair}(us, vs),\ \mathsf{leq}(y, x) \twoheadrightarrow \mathsf{false}
\end{aligned}$$

In recent years automatic tools that try to show various properties of (C)TRSs have been developed. Since these tools are complex pieces of software, wrong answers are not unheard of. To increase their trustworthiness a common approach is to formalize the techniques used in these tools on the computer using a proof assistant.[1] From such a formalization we then obtain a formally verified program that can certify the output of automated tools.

One of the largest efforts in this respect is the IsaFoR/CeTA[2] project. The ***I**sabelle **F**ormalization of **R**ewriting* (or IsaFoR for short) is developed in the proof assistant Isabelle/HOL employing *Higher-Order Logic*. Before we can formally certify the output of automated tools, we have to formalize all of the underlying theory. In Isabelle/HOL it is possible to generate code [15] for a certifier from a formalization provided that check functions for all techniques and corresponding soundness lemmas have first been formalized. The certifier which is code generated from IsaFoR is

[1] In the following, whenever we use the words "formalize" or "formalization," we mean a computer aided formalization using a proof assistant.

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

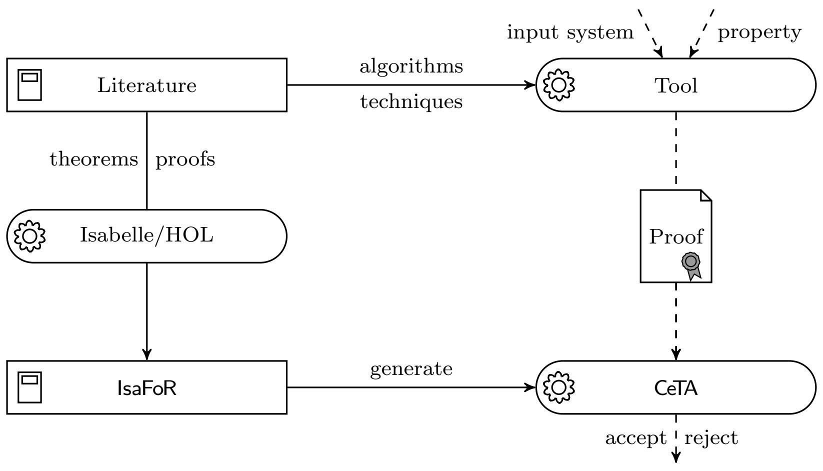


Fig. 1: The certification approach for automated tools.

called CeTA (short for ***Ce**rtified **T**ool **A**ssertions*) and is a fully verified, stand-alone Haskell program. Figure 1 gives a schematic overview of the general approach: We first search the literature for results on the properties we want to prove. Then we formalize the corresponding theorems and proofs in a proof assistant like Isabelle/HOL. Most of the time, this is not straightforward: Sometimes proofs in the literature contain real errors and we have to fix them before we can proceed. Some other time the authors built their proofs on some implicit assumptions that we have to state explicitly in order for the proof to succeed. Almost always there are gaps in the proofs, missing details or tedious technicalities that one may skip on paper and still convince an informed reader about the correctness of the proof but that are needed to convince the computer, that is, Isabelle/HOL.

Then the certifier itself is specified and proved correct inside the proof assistant to allow us to automatically code generate it from IsaFoR—the formal library that is the result of our formalization work.

In parallel we implement the algorithms and techniques we found in the literature in an automatic tool and adapt its output in order for the certifier to be able to parse it. A tool has to provide detailed information about every technique it employed in a proof and state this in form of a *certificate* that is readable by CeTA.

Finally, we arrive at an automatic and reliable two-stage approach to check properties of the input system:

1. The tool tries to decide the desired property of a given input system automatically.
2. If a certificate is generated it is given to CeTA which in turn checks all the involved steps and accepts the certificate if it is correct or rejects it otherwise (giving a hopefully helpful error message).

Of course in general there are various different tools for a certain property, all using similar techniques. The main advantage of the certification approach is that instead of having to prove each of these tools correct independently, we just provide one certifier that is able to rigorously assure correctness of the tools' output with respect to a given input and we only have to prove correctness of this certifier once and for all.

For termination and confluence of plain TRSs, CeTA can handle more than 80% of all generated certificates in the respective tool competitions [10, 23]. However, for many applications plain TRSs are either inconvenient or not expressible enough, leading to several extensions of the base formalism. The one we are interested in here is *conditional term rewriting*. Two prominent areas where conditional rewriting is employed are the rewriting engines of modern proof assistants (like Isabelle's simplifier [25]) and functional(-logic) programming with where-clauses (like Haskell [20] and Curry [1]). An important property of conditional term rewrite systems (CTRSs) is confluence and recently interest in automatic tools to show this property has been growing (see for example the conditional category of the confluence competition [23]).

*Contribution.* Our goal is to get the same coverage as for termination and confluence of plain term rewriting for confluence of conditional term rewriting. So we need a formalization against which we can verify the output of the automatic confluence tools for conditional term rewriting. To this end we have extended IsaFoR by several results from the conditional term rewriting and tree automata literature as well as related topics. More concretely, our contributions are as follows:

- We start with a formalization of orthogonality results for CTRSs (Section 4).
- Then, we present the formalization of a critical pair criterion (Section 5).
- We also give methods for finding non-confluence witnesses (Section 6).
- Moreover, we discuss three methods that help us to ignore certain cases of the confluence analysis (Section 7) by identifying so called *infeasible* conditional critical pairs: the symbol transition graph, decomposition of reachability problems, and tree automata techniques.
- We introduce two supporting methods that allow us to ignore or simplify rules of CTRSs (Section 8).
- Finally (Section 9), we evaluate our approach with extensive experiments.

To facilitate our experiments we have implemented all the formalized methods in the automatic tool ConCon that is able to check confluence (and some other properties) of programs represented as CTRSs. ConCon is taking part in the annual confluence competition—CoCo.[3] There automatic confluence tools test their mettle in several different categories. The problems that these tools try to solve come from the confluence problems database—Cops[4] (version 1137 at the time of writing; where the version of Cops is the number of problems that are contained in that version). Beyond the results we get from this competition we provide extensive experiments, comparing the different methods of ConCon to each other. Certification is really a joint effort by the tool that outputs a proof and the certifier that reads it. For that reason we extended both ConCon and IsaFoR in such a way that the certifier CeTA is now able to check all of the above mentioned techniques and at the same time ConCon can provide detailed enough output for all methods it implements in a format readable by CeTA. Concerning check functions in our formalization it is worth mentioning that their return type is only "morally" bool. In

[3] http://project-coco.uibk.ac.at

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

order to have nice error messages we actually employ a monad. So whenever we need to handle the result of a check function as bool we encapsulate it in a call to *isOK* which results in *False* if there was an error and *True*, otherwise.

This work extends and unifies our previous work [30, 31, 37, 39]. Additionally, we added many examples to clarify the presented results.

In the remainder we give hyperlinks (marked by ☑) to an HTML rendering of our formalizations.

## 2 Preliminaries

While we try to be self-contained, some familiarity with the basics of (conditional) term rewriting [4, 28] will be helpful. We recapitulate terminology and notation that we use in the remainder in a condensed form.

An *abstract rewrite system* (ARS for short) $\mathcal{A}$ is a carrier $A$ together with a binary relation $\to_{\mathcal{A}}$ on $A$. If $\mathcal{A}$ is clear from context we just write $\to$. Instead of the more common $(a, b) \in \to$ we write $a \to b$ and we call this a *rewrite step*. Given an arbitrary binary relation $\to_\alpha$, we write ${}_\alpha\!\leftarrow$, $\to_\alpha^+$, and $\to_\alpha^*$ for its *inverse*, its *transitive closure*, and its *reflexive transitive closure*, respectively. Moreover, given another arbitrary binary relation $\to_\beta$, the relations ${}^*_\alpha\!\leftarrow \cdot \to^*_\beta$ and $\to^*_\beta \cdot {}^*_\alpha\!\leftarrow$ are called *meetability* and *joinability*. We may abbreviate the relation $\to^*_\beta \cdot {}^*_\alpha\!\leftarrow$ by $\downarrow$. Sometimes we call a situation $b\ {}^*_\alpha\!\leftarrow a \to^*_\beta c$ a *diverging situation* or a *peak* if it consists of two single steps $b\ {}_\alpha\!\leftarrow a \to_\beta c$. We say that $\to_\alpha$ has the *diamond property* whenever ${}_\alpha\!\leftarrow \cdot \to_\alpha \subseteq \to_\alpha \cdot {}_\alpha\!\leftarrow$ holds. We say that $\to_\alpha$ and $\to_\beta$ *commute* whenever ${}^*_\alpha\!\leftarrow \cdot \to^*_\beta \subseteq \to^*_\beta \cdot {}^*_\alpha\!\leftarrow$ holds. The same property is called confluence, in case $\alpha$ and $\beta$ coincide. If we have $b \downarrow c$ whenever $b\ {}_\alpha\!\leftarrow a \to_\alpha c$ we call $\to_\alpha$ *locally confluent*. If $\to_\alpha$ has the diamond property and the inclusion $\to_\beta \subseteq \to_\alpha \subseteq \to^*_\beta$ holds then $\to_\beta$ is confluent. An ARS $\mathcal{A}$ is *terminating* if there is no $a \in A$ that admits an infinite rewrite sequence starting from $a$. The following famous result by Newman [24] establishes a connection between termination and confluence: Every terminating and locally confluent ARS is confluent.

We use $\mathcal{V}(\cdot)$ to denote the set of variables occurring in a given list of syntactic objects, like terms, rules, etc. The set of terms $\mathcal{T}(\mathcal{F}, \mathcal{V})$ over a given signature of function symbols $\mathcal{F}$ and set of variables $\mathcal{V}$ is defined inductively: $x \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ for all variables $x \in \mathcal{V}$, and for every $n$-ary function symbol $f \in \mathcal{F}$ and terms $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ also $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. A term is called *linear* if it does not contain multiple occurrences of the same variable. Furthermore, we say that a term $t$ is *ground* if $\mathcal{V}(t) = \varnothing$. The root position of a term is denoted by $\epsilon$. Given a term $t$, we write $\mathcal{P}\mathsf{os}(t)$ for the *set of positions* in $t$ and $t|_p$ with $p \in \mathcal{P}\mathsf{os}(t)$ for the subterm of $t$ at position $p$. We write $s[t]_p$ for the result of replacing $s|_p$ by $t$ in $s$. Given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and a fresh constant $\bot \notin \mathcal{F}$ the function $\mathsf{root}$ that returns the *root symbol* of $t$ is defined as $\mathsf{root}(t) = f$ if $t = f(t_1, \ldots, t_n)$ and $\bot$ otherwise. Although the use of $\bot$ in $\mathsf{root}$ is unusual, it will be helpful in Section 7.2.

Given a fresh constant $\Box$ the terms in the subset of $\mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{V})$ where there is exactly one occurrence of $\Box$ are called *contexts*. We denote the set of all contexts by $\mathcal{C}(\mathcal{F}, \mathcal{V})$ and we also call $\Box$ the *empty context*. If $C$ is a context, $p$ the position of $\Box$ in the context, and $t$ some term then $C[t]$ denotes the term $C[t]_p$. A binary relation $>$ on terms is *closed under contexts* if $C[s] > C[t]$ for all contexts $C$ and terms $s, t$ where $s > t$. A *rewrite rule* is a pair of terms written $\ell \to r$ where the left-hand side $\ell$ is not a variable and there are no variables in $r$ that do not already occur in $\ell$. A rule with variable right-hand side is called *collapsing.* We sometimes label rewrite rules like $\rho\colon \ell \to r$. A set of rewrite rules is called a *term rewrite system (TRS).* A TRS is called *(left-, right-)linear* if all (left-hand, right-hand side) terms are linear. Sometimes we use *extended TRSs* where we only impose the first variable restriction. As usual $\mathcal{R}^{-1}$ denotes the *inverse* of a TRS $\mathcal{R}$. Note that the inverse of a TRS could very well have variable left-hand sides. Given a TRS $\mathcal{R}$ over signature $\mathcal{F}$ the set $\mathcal{D} = \{\mathsf{root}(\ell) \mid \ell \to r \in \mathcal{R}\}$ is called the *defined symbols* of $\mathcal{R}$. In contrast the set $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$ is called the *constructor symbols* (or just the constructors) of $\mathcal{R}$. A term over $\mathcal{T}(\mathcal{C}, \mathcal{V})$ is called a *constructor term.* A *substitution* is a mapping from variables to terms where only finitely many variables are not mapped to themselves. The *empty substitution* is the identity on $\mathcal{V}$ and written $\varepsilon$. We write $t\sigma$ to denote the application of the substitution $\sigma$ to the term $t$ that is defined recursively as $t\sigma = \sigma(t)$ if $t$ is a variable and $t\sigma = f(t_1\sigma, \dots, t_n\sigma)$ if $t = f(t_1, \dots, t_n)$. Sometimes it is useful to represent a substitution by its set of variable bindings $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$. We call a bijective variable substitution $\pi : \mathcal{V} \to \mathcal{V}$ a *variable renaming* or *(variable) permutation*, and denote its *inverse* by $\pi^-$. For two substitutions $\sigma$, $\tau$ and a set of variables $V$ we write $\sigma = \tau\ [V]$ if $\sigma(x) = \tau(x)$ for all $x \in V$. We write $\sigma\tau$ for the *composition* of the two substitutions $\sigma$ and $\tau$ which is defined to be $(\sigma\tau)(x) = \sigma(x)\tau$, that is, a composition lists substitutions in their order of application. Finally, a substitution $\sigma$ is called a *ground substitution* if it maps variables to ground terms. A binary relation $>$ on terms is *closed under substitutions* if $s\sigma > t\sigma$ for all substitutions $\sigma$ and terms $s, t$ where $s > t$. A term $s$ *matches* a term $t$ if $t = s\sigma$ for some substitution $\sigma$. We say that $t$ is an *instance* of $s$ and conversely that $s$ is a *generalization* of $t$. We say that terms $s$ and $t$ *unify*, written $s \sim t$, if $s\sigma = t\sigma$ for some substitution $\sigma$. A binary relation on terms that is closed under contexts and substitutions is called a *rewrite relation.* Given a TRS $\mathcal{R}$ we write $s \to t$ if there exists a rewrite rule $\ell \to r$ in $\mathcal{R}$, a substitution $\sigma$ and a context $C$ such that $s = C[\ell\sigma]$ and $t = C[r\sigma]$. We call the subterm $\ell\sigma$ the *redex.* Given a TRS $\mathcal{R}$ a term $t$ such that there is no step $t \to_{\mathcal{R}} s$ for any term $s$ is called an *$\mathcal{R}$-normal form.* A substitution $\sigma$ is *$\mathcal{R}$-normalized* if $\sigma(x)$ is an $\mathcal{R}$-normal form for all variables $x$. An *oriented conditional term rewrite system (CTRS)* $\mathcal{R}$ is a set of conditional rewrite rules of the shape $\ell \to r \Leftarrow c$ where $\ell$ and $r$ are terms, $\ell$ is not a variable, and $c$ is a possibly empty sequence of pairs of terms (called *conditions*) $s_1 \twoheadrightarrow t_1, \dots, s_k \twoheadrightarrow t_k$. The extended TRS obtained from a CTRS $\mathcal{R}$ by dropping the conditional parts of the rewrite rules is denoted by $\mathcal{R}_\mathsf{u}$ and called the *underlying TRS* of $\mathcal{R}$. Note that $\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}_u}$. For a rule $\rho : \ell \to r \Leftarrow c$ of a CTRS $\mathcal{R}$ the set of *extra variables* is defined as $\mathcal{EV}(\rho) = \mathcal{V}(\rho) - \mathcal{V}(\ell)$. The rewrite relation induced by a CTRS $\mathcal{R}$ is structured into *levels.* For each level $i$, a TRS $\mathcal{R}_i$ is defined recursively by

$$\mathcal{R}_0 = \varnothing$$
$$\mathcal{R}_{i+1} = \{\ell\sigma \to r\sigma \mid \ell \to r \Leftarrow c \in \mathcal{R} \text{ and } s\sigma \xrightarrow[\mathcal{R}_i]{*} t\sigma \text{ for all } s \twoheadrightarrow t \in c\}$$

where $\to_{\mathcal{R}_i}$ denotes the rewrite relation of the (unconditional) TRS $\mathcal{R}_i$. We write $s \to_{\mathcal{R},i} t$ (or $s \to_i t$ whenever $\mathcal{R}$ is clear from the context) if we have $s \to_{\mathcal{R}_i} t$. The rewrite relation of $\mathcal{R}$ is defined as $\to_{\mathcal{R}} = \bigcup_{i \geqslant 0} \to_{\mathcal{R}_i}$. Furthermore, we write $\sigma, i \vdash c$ whenever $s\sigma \to_i^* t\sigma$ for all $s \twoheadrightarrow t$ in $c$ and we say that $\sigma$ *satisfies* the set of conditions $c$. If the level $i$ is not important we also write $\sigma \vdash c$. When there is no substitution $\sigma$ such that $\sigma \vdash c$ we say that the set of conditions $c$ is *infeasible*. Note that a *conditional rewrite step* $s \rightarrow_{\mathcal{R}} t$ employing $\ell \rightarrow r \Leftarrow c \in \mathcal{R}$ and substitution $\sigma$ is only possible if $\sigma$ satisfies $c$. A conditional rule $\ell \rightarrow r \Leftarrow c$ is said to be of *type 3* if $\mathcal{V}(r) \subseteq \mathcal{V}(\ell, c)$ and of *type 4* otherwise. A rule of type 3 for which $\mathcal{V}(s_i) \subseteq \mathcal{V}(\ell, t_1, \ldots, t_{i-1})$ holds for all $1 \leqslant i \leqslant k$ is called *deterministic*. A term $t$ is *strongly $\mathcal{R}$-irreducible* if $t\sigma$ is an $\mathcal{R}$-normal form for all $\mathcal{R}$-normalized substitutions $\sigma$. A deterministic rule for which the terms $t_i$ for all $1 \leqslant i \leqslant k$ are strongly $\mathcal{R}$-irreducible is called *strongly deterministic*. If a CTRS only consists of rules of type 3 it is called a *3-CTRS*, if it only consists of deterministic rules it is called a *DCTRS*, and finally if it only consists of strongly deterministic rules it is called *SDCTRS*. Two variable-disjoint variants of rules $\ell_1 \rightarrow r_1 \Leftarrow c_1$ and $\ell_2 \rightarrow r_2 \Leftarrow c_2$ in $\mathcal{R}$ such that $\ell_1|_p$ is not a variable and $\ell_1|_p\mu = \ell_2\mu$ with most general unifier (mgu) $\mu$, constitute a *conditional overlap*. A conditional overlap that does not result from overlapping two variants of the same rule at the root gives rise to a *conditional critical pair* (CCP) $\ell_1\mu[r_2\mu]_p \approx r_1\mu \Leftarrow c_1\mu, c_2\mu$. A (C)TRS $\mathcal{R}$ is *orthogonal* if it is left-linear and it has no (conditional) critical pairs. The following famous result (known as the Critical Pair Lemma [17]) shows why we can concentrate on critical peaks: A TRS is locally confluent if and only if all its critical pairs are joinable. So for a terminating TRS by Newman's Lemma and the Critical Pair Lemma we immediately get the following result: A terminating TRS is confluent if and only if all its critical pairs are joinable. Note that this means that in the unconditional case confluence is decidable for terminating TRSs. A CCP $u \twoheadrightarrow v \Leftarrow c$ is said to be *infeasible* if its conditions are infeasible. Moreover, a CCP is *joinable* if $u\sigma \downarrow_{\mathcal{R}} v\sigma$ for all substitutions $\sigma$ that satisfy $c$. The topmost part of a term that does not change under rewriting (sometimes called its "cap") can be approximated for example by the $\mathsf{tcap}$ function [11]. Informally, $\mathsf{tcap}(x)$ for a variable $x$ results in a fresh variable, while $\mathsf{tcap}(t)$ for a non-variable term $t = f(t_1, \ldots, t_n)$ is obtained by recursively computing $u = f(\mathsf{tcap}(t_1), \ldots, \mathsf{tcap}(t_n))$ and then asserting $\mathsf{tcap}(t) = u$ in case $u$ does not unify with any left-hand side of rules in $\mathcal{R}$, and a fresh variable, otherwise. It is well known that $\mathsf{tcap}(s) \not\sim t$ implies non-reachability of $t$ from $s$. For two terms $s$ and $t$ we write $s \mathrel{+\!\!\!\!\twoheadrightarrow}_{\mathcal{R}} t$ if $s = t$, $s \rightarrow_{\mathcal{R}} t$, or $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, and $s_i \mathrel{+\!\!\!\!\twoheadrightarrow}_{\mathcal{R}} t_i$ for all $1 \leqslant i \leqslant n$. The relation $\mathrel{+\!\!\!\!\twoheadrightarrow}_{\mathcal{R}}$ is called *parallel rewriting*. The well-known Parallel Moves Lemma states that for every orthogonal TRS $\mathcal{R}$ its parallel rewrite relation $\mathrel{+\!\!\!\!\twoheadrightarrow}_{\mathcal{R}}$ has the diamond property. Together with the well-known fact, that $\rightarrow \subseteq \mathrel{+\!\!\!\!\twoheadrightarrow} \subseteq \rightarrow^*$, we get that orthogonal TRSs are confluent. A CTRS $\mathcal{R}$ over signature $\mathcal{F}$ is called *level-commuting* if for all levels $m, n$ and terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ whenever $s \;{}_m^*{\leftarrow} \cdot \rightarrow_n^* t$ also $s \rightarrow_n^* \cdot {}_m^*{\leftarrow}\; t$. If $n = m$ this property is called *level-confluence* of a CTRS. Level-commutation implies level-confluence which in turn implies confluence. We denote the proper superterm relation by $\rhd$ and define $\succ_{\mathsf{st}} = (\succ \cup \rhd)^+$ for any order $\succ$. A DCTRS $\mathcal{R}$ over signature $\mathcal{F}$ is *quasi-reductive* if there is an extension $\mathcal{F}'$ of the signature (that is, $\mathcal{F} \subseteq \mathcal{F}'$) and a well-founded partial order $\succ$ on $\mathcal{T}(\mathcal{F}', \mathcal{V})$ that is closed under contexts such that for every substitution $\sigma\colon \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}', \mathcal{V})$ and every rule $\ell \rightarrow r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k$ in $\mathcal{R}$ we have that $s_j\sigma \succeq t_j\sigma$ for $1 \leqslant j < i$ implies $\ell\sigma \succ_{\mathsf{st}} s_i\sigma$ for all $1 \leqslant i \leqslant k$, and $s_j\sigma \succeq t_j\sigma$ for $1 \leqslant j \leqslant k$ implies $\ell\sigma \succ r\sigma$. A DCTRS $\mathcal{R}$ over signature $\mathcal{F}$ is *quasi-decreasing* if there is a well-founded order $\succ$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $\succ = \succ_{\mathsf{st}}$, $\rightarrow_{\mathcal{R}} \subseteq \succ$, and for all rules $\ell \rightarrow r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k$ in $\mathcal{R}$, all substitutions $\sigma\colon \mathcal{V} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V})$, and all $1 \leqslant i \leqslant k$, if $s_j\sigma \rightarrow_{\mathcal{R}}^* t_j\sigma$ for all $1 \leqslant j < i$ then $\ell\sigma \succ s_i\sigma$. Quasi-reductivity implies quasi-decreasingness—a fact that is available in IsaFoR.

A bottom-up non-deterministic finite *tree automaton (TA)* $\mathcal{A} = \langle \mathcal{F}, Q, Q_f, \Delta \rangle$ consists of four parts: a signature $\mathcal{F}$, a set of *states* $Q$ disjoint from $\mathcal{F}$, a set of *final states* $Q_f \subseteq Q$, and a set of *transitions* $\Delta$ of the shape $f(q_1, \ldots, q_n) \to q$ with $f/n \in \mathcal{F}$ and $q_1, \ldots, q_n, q \in Q$ or $q \to p$ with $q, p \in Q$. The *language* of a TA $\mathcal{A}$ is given by the set $L(\mathcal{A}) = \{t \in \mathcal{T}(\mathcal{F}) \mid \text{there is a } q \in Q_f \text{ such that } t \to_{\mathcal{A}}^* q\}$. We say that a set of ground terms $E$ is *regular* (or a *regular language*) if there is a TA $\mathcal{A}$ such that $L(\mathcal{A}) = E$. To represent a TA we usually only need to specify the transitions and mark the final states. The signature and the set of states is clear from the transitions. A substitution from variables to states is called a *state substitution.*

## 3 Roadmap of Formalized Methods

This section presents an overview of all confluence methods for CTRSs that are available in CeTA. There are basically three ways to show confluence of a CTRS:

1. use a transformation to reduce the problem to the unconditional case,
2. exploit orthogonality, or
3. employ a critical pair criterion.

All of them are supported by CeTA.

The second and third approaches among other things also analyze the critical pairs of a CTRS. Being able to ignore certain critical pairs simplifies this analysis. Now, if the conditions of a CCP are not satisfiable then the resulting equation can never be utilized and hence is harmless for the confluence of the CTRS under consideration. CCPs with unsatisfiable conditions are called *infeasible* and we can safely ignore them for the purpose of confluence. Our certifier CeTA supports the following reachability analysis based methods to show infeasibility of CCPs:

1. unification,
2. symbol transition graph analysis,
3. decomposition of reachability problems, and
4. tree automata completion.

When checking for confluence we are not only interested in positive answers. If a CTRS is *not* confluent we have to find a witness to prove it. This is also supported by CeTA.

Sometimes the above methods for (non-)confluence are not directly applicable. CeTA supports two sound methods that can "simplify" a given CTRS and make it amenable for them. The first one removes rules with infeasible conditions (because they do not influence the rewrite relation anyway), while the second one "reshapes" certain conditional rewrite rules in such a way that other methods become more applicable, without changing the induced rewrite relation.

*Confluence Methods.* Concerning the reduction to the unconditional case the following theorem is based on work by Nishida et al. [27] and has been formalized by Winkler and Thiemann [43, Theorem 20].

**Theorem 1** *For a DCTRS $\mathcal{R}$ and a source preserving unraveling $\mathsf{U}$ if the TRS $\mathsf{U}(\mathcal{R})$ is left-linear and confluent then $\mathcal{R}$ is confluent.* □

Since this result is not part of our formalization effort we refer the interested reader to the work of Winkler and Thiemann [43] for further details.

The first orthogonality result for 3-CTRSs is due to Suzuki et al. [40, Corollary 4.7].

**Theorem 2** *Almost orthogonal, extended properly oriented, right-stable, and oriented 3-CTRSs are confluent.* □

We slightly extend the original result and formalize it. This formalization is the topic of Section 4.

A critical pair criterion for SDCTRSs was published by Avenhaus and Loría-Sáenz [3, Theorem 4.1].

**Theorem 3** *Let the SDCTRS $\mathcal{R}$ be quasi-decreasing. Then $\mathcal{R}$ is confluent iff all CCPs are joinable.* □

Again we slightly extend this result and formalize it. A textual description of this formalization is given in Section 5.

*Non-Confluence Methods.* The following non-confluence result directly follows from the definition of confluence:

**Lemma 1** *Given a CTRS $\mathcal{R}$ if we find a diverging situation $t \;{}_{\mathcal{R}}^{+}\!\!\leftarrow s \rightarrow_{\mathcal{R}}^{+} u$ where $t$ and $u$ are not joinable then $\mathcal{R}$ is non-confluent.* □

Further details are presented in Section 6.

*Infeasibility and Supporting Methods.* Infeasibility of conditions can be exploited for several subtasks when trying to decide confluence of a CTRS:

1. rules with infeasible conditions can be dropped from a CTRS,
2. CCPs with infeasible conditions are trivially joinable, and finally,
3. CCPs between infeasible rules can be ignored for the purpose of orthogonality.

The details on how exactly this works in CeTA are given in Sections 7 and 8. Finally, inlining of conditions (which is also explained in detail in Section 8) is inspired by inlining of let-constructs and where-expressions in compilers. In practice, exhaustive application of inlining increases the applicability of other methods like infeasibility via unification and non-confluence via plain rewriting.

In the next section we start by showing how to extend the well-known orthogonality criterion from the unconditional to the conditional case.

## 4 Orthogonality

Unlike orthogonal TRSs, orthogonal CTRSs are not confluent in general, as witnessed by Cops #524 (which is similar to an example by Ida and Okui [18]):

*Example 1 (Cops #524)* Consider the 3-CTRS consisting of the three rules:

$$\mathsf{a} \to x \Leftarrow \mathsf{f}(x) \twoheadrightarrow \mathsf{k} \qquad\qquad \mathsf{f}(\mathsf{b}) \to \mathsf{k} \qquad\qquad \mathsf{f}(\mathsf{c}) \to \mathsf{k}$$

It is orthogonal but not confluent, since we have the peak $\mathsf{b} \leftarrow \mathsf{a} \to \mathsf{c}$ but the terms $\mathsf{b}$ and $\mathsf{c}$ are both normal forms and thus not joinable.

$$\begin{tikzcd}
 & \mathsf{f}(\mathsf{a}) \arrow[dl, "1"'] \arrow[dr, "2"] & \\
\mathsf{a} \arrow[r, "1"'] & \mathsf{b} \arrow[r, "1"'] & \mathsf{c}
\end{tikzcd}$$

Fig. 2: For CTRSs parallel rewriting does not have the diamond property.

We have to impose further restrictions on the distribution of variables in the rules such that during matching the substitution for the rewrite step can be built in a deterministic way. To this end we use the following notions due to Suzuki et al. [40].

**Definition 1 (Right-stability, proper orientedness)** A conditional rewrite rule $\ell \to r \Leftarrow c$ with $k$ conditions $c = s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k$ is called

– *right-stable* whenever we have $\mathcal{V}(t_i) \cap \mathcal{V}(\ell, c_{i-1}, s_i) = \varnothing$ and $t_i$ is either a linear constructor term or a ground $\mathcal{R}_u$-normal form, for all $1 \leqslant i \leqslant k$; and
– *properly oriented* if whenever $\mathcal{V}(r) \not\subseteq \mathcal{V}(\ell)$ then $\mathcal{V}(s_i) \subseteq \mathcal{V}(\ell, t_1, \ldots, t_{i-1})$ for all $1 \leqslant i \leqslant k$.

A CTRS consisting solely of right-stable rules is called *right-stable*. Likewise, a CTRS only containing properly oriented rules is called *properly oriented*.

Note that proper orientedness is really just a relaxation of determinism in that we only demand determinism if there are extra variables in right-hand sides of rules and not for all rules (see Section 2). Now the class of CTRSs we are targeting are orthogonal, properly oriented, right-stable, and oriented 3-CTRSs. Remember that in the unconditional case we employ the Parallel Moves Lemma to show confluence of orthogonal systems. So for a CTRS $\mathcal{R}$ we write $s \mathrel{+\!\!\!\!\twoheadrightarrow}_n t$ if $t$ can be obtained from $s$ by contracting a set of pairwise disjoint redexes in $s$ using $\mathcal{R}_n$. Unfortunately, the Parallel Moves Lemma does not hold for our class of CTRSs as witnessed by Cops #334 [40, Example 4.4].

*Example 2 (Cops #334)* Consider the orthogonal, properly oriented, right-stable, and oriented 3-CTRS consisting of the three rules:

$$\mathsf{f}(x) \to y \Leftarrow x \twoheadrightarrow y \qquad\qquad \mathsf{a} \to \mathsf{b} \qquad\qquad \mathsf{b} \to \mathsf{c}$$

In the peak depicted in Figure 2 no parallel rewrite step from $\mathsf{a}$ to $\mathsf{c}$ is possible, but we still can rewrite $\mathsf{a}$ to the normal form $\mathsf{c}$ with a sequence whose level is smaller than the level of the step from $\mathsf{f}(\mathsf{a})$ to $\mathsf{c}$. Incorporating these findings into parallel rewriting for CTRSs we arrive at the notion of *extended parallel rewriting*:

**Definition 2 (Extended parallel rewriting)** First we adopt the convention that the number of holes of a multi-hole context is denoted by the corresponding lower-case letter, for example, $c$ for $C$, $d$ for $D$, $e$ for $E$ etc. Then we say that there is an *extended parallel rewrite step at level* $n$ from $s$ to $t$, written $s \mathrel{+\!\!\!\!+\!\!\!\!\twoheadrightarrow}_n t$, whenever we have a multi-hole context $C$, and sequences of terms $s_1, \ldots, s_c$ and $t_1, \ldots, t_c$, such that $s = C[s_1, \ldots, s_c]$, $t = C[t_1, \ldots, t_c]$, and for all $1 \leqslant i \leqslant k$ we have either

1. $(s_i, t_i) \in \mathcal{R}_n$ (that is, a root step at level $n$), or

2. $s_i \to_{n-1}^* t_i$.

It is easy to see that $\to_n \subseteq \mathrel{+\!\!\!\!\twoheadrightarrow}_n \subseteq \to_n^*$. We are ready to state the following variation of the Parallel Moves Lemma.

**Theorem 4** *For orthogonal, properly oriented, right-stable, and oriented 3-CTRSs extended parallel rewriting has the commuting diamond property.* ☐

Because the commuting diamond property obviously implies level-commutation the above theorem yields the following confluence result.

**Corollary 1** *Orthogonal, properly oriented, right-stable, and oriented 3-CTRSs are confluent.* ☐

We can further improve upon this result by using a looser notion of proper orientedness.

**Definition 3 (Extended proper orientedness ☑)** A conditional rule $\ell \to r \Leftarrow c$ with $k$ conditions $c = s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k$ is called *extended properly oriented* when either $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ or there is some $0 \leqslant m \leqslant k$ such that $\mathcal{V}(s_i) \subseteq \mathcal{V}(\ell, t_1, \ldots, t_{i-1})$ for all $1 \leqslant i \leqslant m$ and $\mathcal{V}(r) \cap \mathcal{V}(s_j \twoheadrightarrow t_j) \subseteq \mathcal{V}(\ell, t_1, \ldots, t_m)$ for all $m < j \leqslant k$. A CTRS only containing extended properly oriented rules is called an *extended properly oriented* CTRS.

There are CTRSs that are extended properly oriented but not properly oriented as witnessed by Cops #548:

*Example 3 (Cops #548)* Consider the oriented 3-CTRS consisting of the single rule

$$\mathsf{g}(x) \to y \Leftarrow x \twoheadrightarrow y,\ z \twoheadrightarrow \mathsf{a}$$

It is orthogonal, right-stable, and extended properly oriented but *not* properly oriented, because of the extra variable $z$ in the second condition.

Observe the following property of a conditional rule $\ell \to r \Leftarrow c$ of type 3 with $k$ conditions

$$\text{for some } 0 \leqslant m \leqslant k.\, \mathcal{V}(r) \subseteq \mathcal{V}(\ell, c_m) \cup (\mathcal{V}(r) \cap \mathcal{V}(c_{m+1,k})) \qquad (\star)$$

which we will use later and which directly follows from $\mathcal{V}(r) \subseteq \mathcal{V}(\ell, c)$. Moreover, we additionally loosen the orthogonality restriction to allow overlaps that are harmless.

**Definition 4 (Almost orthogonality modulo infeasibility ☑)** A left-linear CTRS $\mathcal{R}$ is *almost orthogonal (modulo infeasibility)* if each overlap between rules $\ell_1 \to r_1 \Leftarrow c_1$ and $\ell_2 \to r_2 \Leftarrow c_2$ with mgu $\mu$ at position $p$ either

1. results from overlapping two variants of the same rule at the root, or
2. is trivial (that is, $p = \epsilon$ and $r_1\mu = r_2\mu$), or
3. is infeasible in the following sense: for arbitrary $m$ and $n$, whenever levels $m$ and $n$ commute, then it is impossible to satisfy the conditions stemming from the first rule on level $m$ and at the same time the conditions stemming from the second rule on level $n$. More formally: $\forall m\, n.\, ({}_m^*\!\leftarrow \cdot \to_n^* \subseteq \to_n^* \cdot {}_m^*\!\leftarrow \implies \nexists \sigma.\, \sigma, m \vdash c_1\mu \wedge \sigma, n \vdash c_2\mu)$.

Note that without 2 and 3, Definition 4 corresponds to plain orthogonality. Also note that by dropping 3, Definition 4 reduces to the definition of *almost orthogonality* given by Hanus [16]. In the following, whenever we talk about *almost orthogonality* we mean Definition 4. Observe that the level-commutation assumption of the third alternative in Definition 4 allows us to reduce non-meetability to non-joinability. That this is useful in practice is shown by the following example featuring Cops #547 [30, Example 3].

*Example 4 (Non-meetability via* tcap*, Cops* #547*)* Consider the CTRS consisting of the two rules:

$$\mathsf{f}(x) \to \mathsf{a} \Leftarrow x \twoheadrightarrow \mathsf{a} \qquad\qquad \mathsf{f}(x) \to \mathsf{b} \Leftarrow x \twoheadrightarrow \mathsf{b}$$

It has the critical pair

$$\mathsf{a} \approx \mathsf{b} \Leftarrow x \twoheadrightarrow \mathsf{a},\ x \twoheadrightarrow \mathsf{b}$$

Since $\mathsf{tcap}(\mathsf{cs}(x,x)) = \mathsf{cs}(y,z) \sim \mathsf{cs}(\mathsf{a},\mathsf{b})$, where $\mathsf{cs}$ is a fresh auxiliary function symbol, we cannot conclude infeasibility via non-reachability analysis using tcap. However, $\mathsf{tcap}(\mathsf{a}) = \mathsf{a} \not\sim \mathsf{b} = \mathsf{tcap}(\mathsf{b})$ shows non-joinability of a and b. By Definition 4.3 this shows non-meetability of a and b and thereby infeasibility of the critical pair.

In general it is beneficial to test for non-meetability via non-joinability of conditions with identical left-hand sides (see also Lemma 29). Finally, we are ready to state this new variation of the Parallel Moves Lemma for extended parallel rewriting.

**Theorem 5** *For almost orthogonal, extended properly oriented, right-stable, and oriented 3-CTRSs extended parallel rewriting has the commuting diamond property.* ☑

*Proof* Let $\mathcal{R}$ be a CTRS satisfying all required properties. The commuting diamond property for parallel rewriting states that ${}_m\!\nleftarrow\!\!\!\!\!\nleftarrow \cdot \twoheadrightarrow\!\!\!\!\!\!\shortparallel_n \subseteq \twoheadrightarrow\!\!\!\!\!\!\shortparallel_n \cdot {}_m\!\nleftarrow\!\!\!\!\!\nleftarrow$ for all $m$ and $n$. We proceed by complete induction on $m+n$. By induction hypothesis (IH) we may assume the result for all $m'+n' < m+n$. Now consider the peak $t \;{}_m\!\nleftarrow\!\!\!\!\!\nleftarrow s \twoheadrightarrow\!\!\!\!\!\!\shortparallel_n u$. If any of $m$ and $n$ equals 0, we are done (since $\twoheadrightarrow\!\!\!\!\!\!\shortparallel_0$ is the identity relation). Thus we may assume $m = m'+1$ and $n = n'+1$ for some $m'$ and $n'$. By the definition of extended parallel rewriting, we obtain multihole contexts $C$ and $D$, and sequences of terms $s_1,\ldots,s_c$, $t_1,\ldots,t_c$, $u_1,\ldots,u_d$, $v_1,\ldots,v_d$, such that $s = C[s_1,\ldots,s_c]$ and $t = C[t_1,\ldots,t_c]$, as well as $s = D[u_1,\ldots,u_d]$ and $u = D[v_1,\ldots,v_d]$; and $(s_i,t_i) \in \mathcal{R}_m$ or $s_i \to^*_{m'} t_i$ for all $1 \leqslant i \leqslant c$, as well as $(u_i,v_i) \in \mathcal{R}_n$ or $u_i \to^*_{n'} v_i$ for all $1 \leqslant i \leqslant d$.

It is relatively easy to define the greatest lower bound $C \sqcap D$ of two contexts $C$ and $D$ by a recursive function (that simultaneously traverses the two contexts in a top-down manner and replaces subcontexts that differ by a hole) and prove that we obtain a lower semilattice. Now we identify the common part $E$ of $C$ and $D$, employing the semilattice properties of multihole contexts, that is, $E = C \sqcap D$. Then $C = E[C_1,\ldots,C_e]$ and $D = E[D_1,\ldots,D_e]$ for some multihole contexts $C_1,\ldots,C_e$ and $D_1,\ldots,D_e$ such that for each $1 \leqslant i \leqslant e$ we have $C_i = \square$ or $D_i = \square$. This also means that there is a sequence of terms $s'_1,\ldots,s'_e$ such that $s = E[s'_1,\ldots,s'_e]$ and for all $1 \leqslant i \leqslant e$, we have $s'_i = C_i[s_{k_i},\ldots,s_{k_i+c_i-1}]$ for some subsequence $s_{k_i},\ldots,s_{k_i+c_i-1}$ of $s_1,\ldots,s_c$ (we denote similar terms for $t$, $u$, and $v$ by $t'_i$, $u'_i$, and $v'_i$, respectively). Moreover, note that by construction $s'_i = u'_i$ for all $1 \leqslant i \leqslant e$.

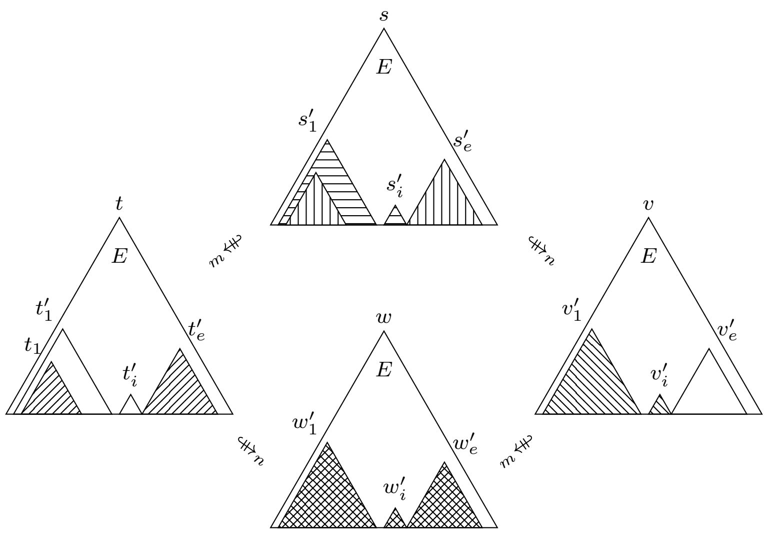


Fig. 3: Commuting diamond property of extended parallel rewriting.

Since extended parallel rewriting is closed under multihole contexts, it suffices to show that for each $1 \leqslant i \leqslant e$ there is a term $v$ such that $t'_i \nrightarrow\!\!\!\!\twoheadrightarrow_n v \;{}_m\!\!\twoheadleftarrow\!\!\!\!\nleftarrow v'_i$, in order to conclude the proof. This is depicted in Figure 3, where $w'_i$ denotes the respective $v$'s. We concentrate on the case $C_i = \square$ (the case $D_i = \square$ is completely symmetric). Moreover, note that when we have $s'_i \rightarrow^*_{m'} t'_i$, the proof concludes by IH (together with some basic properties of the involved relations), and thus we remain with $(s'_i, t'_i) \in \mathcal{R}_m$. At this point we distinguish the following cases:

1. ($D_i = \square$). Also here, the non-root case $u'_i \rightarrow^*_{n'} v'_i$ is covered by the IH. Thus, we may restrict to $(u'_i, v'_i) \in \mathcal{R}_n$, giving rise to a root overlap. Since $\mathcal{R}$ is almost orthogonal, this means that either the resulting conditions are not satisfiable or the resulting terms are the same (in both of these cases we are done), or two variable disjoint variants of the same rule $\ell \rightarrow r \Leftarrow c$ with conditions $c = s_1 \twoheadrightarrow t_1, \dots, s_j \twoheadrightarrow t_j$ were involved, that is, $u'_i = \ell\sigma_1 = \ell\sigma_2$ for some substitutions $\sigma_1$ and $\sigma_2$ that both satisfy all conditions in $c$. Without extra variables in $r$, this is the end of the story (since then $r\sigma_1 = r\sigma_2$); but we also want to cover the case where $\mathcal{V}(r) \nsubseteq \mathcal{V}(\ell)$, and thus have to reason why this does not cause any trouble. Together with the fact that $\ell \rightarrow r \Leftarrow c$ is extended properly oriented we obtain a $0 \leqslant k \leqslant j$ such that
   (a) $\mathcal{V}(s_i) \subseteq \mathcal{V}(\ell, t_1, \dots, t_{i-1})$ for all $1 \leqslant i \leqslant k$ and
   (b) $\mathcal{V}(r) \cap \mathcal{V}(s_i \twoheadrightarrow t_i) \subseteq \mathcal{V}(\ell, t_1, \dots, t_k)$ for all $k < i \leqslant j$
   by Definition 3. Then we prove by an inner induction on $i \leqslant j$ that there is a substitution $\sigma$ such that
   (c) $\sigma(x) = \sigma_1(x) = \sigma_2(x)$ for all $x$ in $\mathcal{V}(\ell)$, and
   (d) $\sigma_1(x) \nrightarrow\!\!\!\!\twoheadrightarrow^*_{n'} \sigma(x)$ and $\sigma_2(x) \nrightarrow\!\!\!\!\twoheadrightarrow^*_{m'} \sigma(x)$ for all $x$ in $\mathcal{V}(\ell, c_{\min\{i,k\}}) \cup (\mathcal{V}(r) \cap \mathcal{V}(c_{k+1,i}))$.
   In the base case $\sigma_1$ satisfies the requirements. So suppose $i > 0$ and assume by IH that both properties hold for $i - 1$ and some substitution $\sigma$. If $i > k$

we are done by 1b. Otherwise $i \leqslant k$. Now consider the condition $s_i \twoheadrightarrow t_i$. By 1a we have $\mathcal{V}(s_i) \subseteq \mathcal{V}(\ell, c_{i-1})$. Using the IH for 1d we obtain $s_i\sigma_1 \mathrel{+\!\!\!\twoheadrightarrow}^*_{n'} s_i\sigma$ and $s_i\sigma_2 \mathrel{+\!\!\!\twoheadrightarrow}^*_{m'} s_i\sigma$. Moreover $s_i\sigma_1 \mathrel{+\!\!\!\twoheadrightarrow}^*_{m'} t_i\sigma_1$ and $s_i\sigma_2 \mathrel{+\!\!\!\twoheadrightarrow}^*_{n'} t_i\sigma_2$ since $\sigma_1$ and $\sigma_2$ satisfy $c$, and thus by the outer IH we obtain $s'$ such that $t_i\sigma_1 \mathrel{+\!\!\!\twoheadrightarrow}^*_{n'} s'$ and $t_i\sigma_2 \mathrel{+\!\!\!\twoheadrightarrow}^*_{m'} s'$. Recall that by right-stability $t_i$ is either a ground $\mathcal{R}_u$-normal form or a linear constructor term. In the former case $t_i\sigma_1 = t_i\sigma_2 = s'$ and hence $\sigma$ satisfies 1c and 1d. In the latter case right-stability allows us to combine the restriction of $\sigma_1$ to $\mathcal{V}(t_i)$ and the restriction of $\sigma$ to $\mathcal{V}(\ell, c_{i-1})$ into a substitution satisfying 1c and 1d. This concludes the inner induction. Since $\mathcal{R}$ is an extended properly oriented 3-CTRS, using ($\star$) together with 1d shows $r\sigma_1 \mathrel{+\!\!\!\twoheadrightarrow}^*_{n'} r\sigma$ and $r\sigma_2 \mathrel{+\!\!\!\twoheadrightarrow}^*_{m'} r\sigma$. Since $\mathrel{+\!\!\!\twoheadrightarrow}^*_{n'} \subseteq \mathrel{+\!\!\!\twoheadrightarrow}_n$ and $\mathrel{+\!\!\!\twoheadrightarrow}^*_{m'} \subseteq \mathrel{+\!\!\!\twoheadrightarrow}_m$ we can take $v = r\sigma$ to conclude this case.

2. ($D_i \neq \square$). Then for some $1 \leqslant k \leqslant d$, we have $(u_j, v_j) \in \mathcal{R}_n$ or $u_j \to^*_{n'} v_j$ for all $k \leqslant j \leqslant k + d_i - 1$, that is, an extended parallel rewrite step of level $n$ from $s'_i = u'_i = D_i[u_{k_i}, \ldots, u_{k_i+d_i-1}]$ to $D_i[v_{k_i}, \ldots, v_{k_i+d_i-1}] = v'_i$. Since $\mathcal{R}$ is almost orthogonal and, by $D_i \neq \square$, root overlaps are excluded, the constituent parts of the extended parallel step from $s'_i$ to $v'_i$ take place exclusively inside the substitution of the root step to the left (which is somewhat obvious but surprisingly hard to formalize, even more so when having to deal with infeasibility). We again close this case by induction on the number of conditions making use of right-stability of $\mathcal{R}$. □

The same reasoning as before immediately yields the main result of this section—Theorem 2. Clearly, applicability of Theorem 2 relies on having powerful techniques for proving infeasibility at our disposal. Those are the topic of Section 7.

### 4.1 Certification

Since all the properties of our target CTRSs are syntactical it is straightforward to implement the corresponding check functions. If there are no critical pairs the certificate only contains one element that states that the *almost orthogonal* criterion was used. The syntactic properties are checked by CeTA. More involved proofs contain subproofs of infeasibility for all the CCPs (see Section 7).

The formalization of the methods described in this section can be found in the following IsaFoR theory files:

```
thys/Conditional_Rewriting/              thys/Conditional_Rewriting/
   Conditional_Rewriting.thy                Level_Confluence.thy
   Conditional_Critical_Pairs.thy           Level_Confluence_Impl.thy
```

The next section explores a method to show confluence of quasi-decreasing CTRSs provided all of their conditional critical pairs are joinable.

## 5 A Critical Pair Criterion

In the previous section we have seen how to adapt the result that orthogonal TRSs are confluent to the conditional case. Here we do the same for terminating and locally confluent TRSs. Remember that for terminating TRSs confluence is decidable. We just have to check if all CPs are joinable. This well-known result of unconditional term rewriting directly follows from Newman's Lemma and the Critical Pair Lemma (see Section 2). Unfortunately, the same result does not hold in the conditional case. To begin with, the Critical Pair Lemma does not hold for CTRSs as witnessed by the following example featuring Cops #269, the oriented version of a join system due to Bergstra and Klop [5, Example 3.6].

*Example 5 (Cops #269)* The CTRS $\mathcal{R}$ consisting of the two rules

$$\mathsf{c}(x) \to \mathsf{e} \Leftarrow x \twoheadrightarrow \mathsf{c}(x) \qquad\qquad \mathsf{b} \to \mathsf{c}(\mathsf{b})$$

has no CCPs at all but is not (locally) confluent due to the peak $\mathsf{c}(\mathsf{e}) \leftarrow \mathsf{c}(\mathsf{c}(\mathsf{b})) \to \mathsf{e}$, where both steps employ the first rule—the one to the left at position 1 and the one to the right at the root position—and the conditions are satisfied by the second rule. Since $\mathsf{c}(\mathsf{e})$ and $\mathsf{e}$ are two different normal forms we have a non-joinable peak and hence a non-confluent system.

The above system is not terminating. Maybe terminating CTRSs where all CCPs are joinable are confluent? The next example featuring Cops #553, the oriented version of a join system due to Dershowitz et al. [6, Example B], shows that this is not the case.

*Example 6 (Cops #553)* Consider the terminating CTRS $\mathcal{R}$:

$$\begin{array}{llllll} \mathsf{c} \to \mathsf{k}(\mathsf{f}(\mathsf{a})) & (1) & \mathsf{h}(x) \to \mathsf{k}(x) & (3) & \mathsf{a} \to \mathsf{b} & (5) \\ \mathsf{c} \to \mathsf{k}(\mathsf{g}(\mathsf{b})) & (2) & \mathsf{h}(\mathsf{f}(\mathsf{a})) \to \mathsf{c} & (4) & \mathsf{f}(x) \to \mathsf{g}(x) \Leftarrow \mathsf{h}(\mathsf{f}(x)) \twoheadrightarrow \mathsf{k}(\mathsf{g}(\mathsf{b})) & (6) \end{array}$$

There are four CCPs (modulo symmetry) that are all joinable as shown in the diagrams below (where the number of the rule used in a step is attached to the corresponding arrow).

$$\begin{tikzcd} & \mathsf{c} \arrow[dl, "1"'] \arrow[dr, "2"] & \\ \mathsf{k}(\mathsf{f}(\mathsf{a})) \arrow[dr, "6"'] & & \mathsf{k}(\mathsf{g}(\mathsf{b})) \\ & \mathsf{k}(\mathsf{g}(\mathsf{a})) \arrow[ur, "5"'] & \end{tikzcd} \qquad \begin{tikzcd} & \mathsf{h}(\mathsf{f}(\mathsf{a})) \arrow[dl, "3"'] \arrow[dr, "4"] & \\ \mathsf{k}(\mathsf{f}(\mathsf{a})) & & \mathsf{c} \arrow[ll, "1"] \end{tikzcd}$$

$$\begin{tikzcd} & \mathsf{h}(\mathsf{f}(\mathsf{a})) \arrow[dl, "5"'] \arrow[dr, "4"] & \\ \mathsf{h}(\mathsf{f}(\mathsf{b})) \arrow[d, "3"'] & & \mathsf{c} \arrow[d, "1"] \\ \mathsf{k}(\mathsf{f}(\mathsf{b})) & & \mathsf{k}(\mathsf{f}(\mathsf{a})) \arrow[ll, "5"] \end{tikzcd} \qquad \begin{tikzcd} & \mathsf{h}(\mathsf{f}(\mathsf{a})) \arrow[dl, "6"'] \arrow[dr, "4"] & \\ \mathsf{h}(\mathsf{g}(\mathsf{a})) \arrow[d, "5"'] & & \mathsf{c} \arrow[d, "2"] \\ \mathsf{h}(\mathsf{g}(\mathsf{b})) \arrow[rr, "3"'] & & \mathsf{k}(\mathsf{g}(\mathsf{b})) \end{tikzcd}$$

We still have the diverging situation $\mathsf{f}(\mathsf{b}) \leftarrow \mathsf{f}(\mathsf{a}) \to \mathsf{g}(\mathsf{a}) \to \mathsf{g}(\mathsf{b})$ where $\mathsf{f}(\mathsf{b})$ and $\mathsf{g}(\mathsf{b})$ are two different normal forms. So $\mathcal{R}$ is not confluent.

Clearly termination is not enough. Thus, we employ the stronger notion of quasi-decreasingness which in addition to termination also ensures that the rewrite relation is effectively computable and that there are no infinite computations in the conditions. Still, this is not yet sufficient. Because of the possibility of extra variables in conditional rules there are problems for confluence that can arise from overlaps of a rule with itself at the root position and even from variable-overlaps; two cases that are not dangerous at all when considering unconditional term rewriting. This will become clearer after looking at the following two examples featuring Cops #262 and #263 due to Avenhaus and Loría-Sáenz [3, Examples 4.1a,b].

*Example 7 (Cops #262)* Consider the quasi-decreasing CTRS $\mathcal{R}$ consisting of the following three rules (where we write $s + t$ instead of $\mathsf{plus}(s, t)$)

$$\mathsf{0} + y \to y \qquad \mathsf{s}(x) + y \to x + \mathsf{s}(y) \qquad \mathsf{f}(x, y) \to z \Leftarrow x + y \twoheadrightarrow z + z'$$

Now look at the (improper) overlap of the last rule with itself. This yields the (improper) CCP $z \approx w \Leftarrow x + y \twoheadrightarrow z + v, x + y \twoheadrightarrow w + u$. Remember that to show joinability of a CCP we have to check whether it is joinable for all satisfying substitutions. Let us see if we can find a satisfying substitution which makes the CCP non-joinable. Now, if we apply the substitution $\sigma$ mapping $x$, $z$, and $u$ to $\mathsf{s}(\mathsf{0})$ and all other variables to $\mathsf{0}$ to the CCP, the result is

$$\mathsf{s}(\mathsf{0}) \approx \mathsf{0} \Leftarrow \mathsf{s}(\mathsf{0}) + \mathsf{0} \twoheadrightarrow \mathsf{s}(\mathsf{0}) + \mathsf{0}, \mathsf{s}(\mathsf{0}) + \mathsf{0} \twoheadrightarrow \mathsf{0} + \mathsf{s}(\mathsf{0})$$

The first condition is trivially satisfied. To satisfy the second one we use the second rule of $\mathcal{R}$. Clearly $\sigma$ satisfies the CCP. But $\mathsf{s}(\mathsf{0})$ and $\mathsf{0}$ are two different normal forms and so the CCP is not joinable. Which in turn means that $\mathcal{R}$ is not confluent.

*Example 8 (Cops #263)* The quasi-decreasing CTRS $\mathcal{R}$ consists of four rules

$$\mathsf{a} \to \mathsf{c} \qquad \mathsf{g}(\mathsf{a}) \to \mathsf{h}(\mathsf{b}) \qquad \mathsf{h}(\mathsf{b}) \to \mathsf{g}(\mathsf{c}) \qquad \mathsf{f}(x) \to z \Leftarrow \mathsf{g}(x) \twoheadrightarrow \mathsf{h}(z)$$

From the variable-overlap between the first and the last rule we get the (improper) CCP $\mathsf{f}(\mathsf{c}) \approx z \Leftarrow \mathsf{g}(\mathsf{a}) \twoheadrightarrow \mathsf{h}(z)$. With $\sigma(z) = \mathsf{b}$ we have found a satisfying substitution that makes the left- and right-hand sides non-joinable.

To counter these problems we have to restrict the placement of extra variables in the conditions severely. To this end we focus our attention on *strongly deterministic* CTRSs in this section.

The main result of Avenhaus and Loría-Sáenz [3, Theorem 4.1] is the basis for Theorem 3.

**Theorem 6** *If all CCPs of a quasi-decreasing SDCTRS are joinable, then it is confluent.*

*Proof* That all CCPs of a CTRS $\mathcal{R}$ (no need for strong determinism or quasi-decreasingness) are joinable if $\mathcal{R}$ is confluent is straightforward. Thus, we concentrate on the other direction: Assume that all critical pairs are joinable. We consider an arbitrary diverging situation $t \; {}_{\mathcal{R}}^{*}\!\!\leftarrow s \to_{\mathcal{R}}^{*} u$ and prove $t \downarrow_{\mathcal{R}} u$ by well-founded induction with respect to $\succ_{\mathsf{st}}$. Here $\succ$ is the order from the definition of quasi-decreasingness.

By the induction hypothesis (IH) we have that for all terms $t_0, t_1, t_2$ such that $s \succ_{\mathsf{st}} t_0$ and $t_1 \; {}_{\mathcal{R}}^{*}\!\!\leftarrow t_0 \to_{\mathcal{R}}^{*} t_2$ there exists a join $t_1 \to_{\mathcal{R}}^{*} \cdot \; {}_{\mathcal{R}}^{*}\!\!\leftarrow t_2$.

If $s = t$ or $s = u$ then $t$ and $u$ are trivially joinable and we are done. So we may assume that the diverging situation contains at least one step in each direction: $t \; {}_{\mathcal{R}}^{*}\!\!\leftarrow t' \; {}_{\mathcal{R}}\!\!\leftarrow s \to_{\mathcal{R}} u' \to_{\mathcal{R}}^{*} u$.

Let us show that $t' \downarrow_{\mathcal{R}} u'$ holds. Then $t \downarrow_{\mathcal{R}} u$ follows by two applications of the induction hypothesis, as shown in the following diagram:

$$\begin{tikzcd}
 & & s \arrow[dl, "\succ_{\mathsf{st}}"'] \arrow[dr, "\succ_{\mathsf{st}}"] & & \\
 & t' \arrow[dl, "*"'] \arrow[dr, "*"] & & u' \arrow[dl, "*"'] \arrow[dr, "*"] & \\
t \arrow[dr, dotted, "*"] & \mathrm{IH} & \cdot \arrow[dl, dotted, "*"'] & & u \arrow[ddll, dotted, "*"] \\
 & \cdot \arrow[dr, dotted, "*"] & \mathrm{IH} & & \\
 & & \cdot & &
\end{tikzcd}$$

Assume that $s = C[\ell_1\sigma_1]_p \to_{\mathcal{R}} C[r_1\sigma_1]_p = t'$ and $s = D[\ell_2\sigma_2]_q \to_{\mathcal{R}} D[r_2\sigma_2]_q = u'$ for rules $\rho_1 : \ell_1 \to r_1 \Leftarrow c_1$ and $\rho_2 : \ell_2 \to r_2 \Leftarrow c_2$ in $\mathcal{R}$, contexts $C$ and $D$, positions $p$ and $q$, and substitutions $\sigma_1$ and $\sigma_2$ such that $u\sigma_1 \to^*_{\mathcal{R}} v\sigma_1$ for all $u \twoheadrightarrow v \in c_1$ and $u\sigma_2 \to^*_{\mathcal{R}} v\sigma_2$ for all $u \twoheadrightarrow v \in c_2$. There are three possibilities: either the positions are parallel ($p \parallel q$), or $p$ is above $q$ ($p \leqslant q$), or $q$ is above $p$ ($q \leqslant p$). In the first case $t' \downarrow_{\mathcal{R}} u'$ holds because the two redexes do not interfere. The other two cases are symmetric and we only consider $p \leqslant q$ here. If $s \rhd s|_p = \ell_1\sigma_1$ then $s \succ_{\mathsf{st}} \ell_1\sigma_1$ (by definition of $\succ_{\mathsf{st}}$) and there exists a position $r$ such that $q = pr$ and so we have the diverging situation $r_1\sigma_1 \; {}_{\mathcal{R}}^{*}\!\leftarrow \ell_1\sigma_1 \to^*_{\mathcal{R}} \ell_1\sigma_1[r_2\sigma_2]_r$ which is joinable by the induction hypothesis. But then the diverging situation $t' = s[r_1\sigma_1]_p \; {}_{\mathcal{R}}^{*}\!\leftarrow s[\ell_1\sigma_1]_p \to^*_{\mathcal{R}} s[\ell_1\sigma_1[r_2\sigma_2]_r]_q = u'$ is also joinable (by closure under contexts) and we are done. So we may assume that $p = \epsilon$ and thus $s = \ell_1\sigma_1$. Now, either $q$ is a function position in $\ell_1$ or there exists a variable position $q'$ in $\ell_1$ such that $q' \leqslant q$. In the first case we either have

1. a conditional critical pair which is joinable by assumption or we have
2. a root-overlap of variants of the same rule. Unlike in the unconditional case this could lead to non-joinability of the ensuing critical pair because of the extra variables in the right-hand sides of conditional rules. We have $\rho_1\pi = \rho_2$ for some permutation $\pi$. Moreover, $s = \ell_1\sigma_1 = \ell_2\sigma_2$ and we have

$$\pi^{-}\sigma_1 = \sigma_2 \; [\mathcal{V}(\ell_2)] \tag{1}$$

We prove $x\pi^{-}\sigma_1 \downarrow_{\mathcal{R}} x\sigma_2$ for all $x$ in $\mathcal{V}(\rho_2)$. Since $t' = r_1\sigma_1 = r_2\pi^{-}\sigma_1$ and $u' = r_2\sigma_2$ this shows $t' \downarrow_{\mathcal{R}} u'$. Because $\mathcal{R}$ is terminating (by quasi-decreasingness) we may define two normalized substitutions $\sigma'_i$ such that

$$x\pi^{-}\sigma_1 \xrightarrow[\mathcal{R}]{*} x\sigma'_1 \text{ and } x\sigma_2 \xrightarrow[\mathcal{R}]{*} x\sigma'_2 \text{ for all variables } x. \tag{2}$$

We prove $x\sigma'_1 = x\sigma'_2$ for $x \in \mathcal{EV}(\rho_2)$ by an inner induction on the length $k$ of the conditions $c_2 = s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k$. If $\rho_2$ has no conditions this holds vacuously because there are no extra variables. In the step case the inner induction hypothesis is that $x\sigma'_1 = x\sigma'_2$ for $x \in \mathcal{V}(s_1, t_1, \ldots, s_i, t_i) - \mathcal{V}(\ell_2)$ and we have to show that $x\sigma'_1 = x\sigma'_2$ for $x \in \mathcal{V}(s_1, t_1, \ldots, s_{i+1}, t_{i+1}) - \mathcal{V}(\ell_2)$. If $x \in \mathcal{V}(s_1, t_1, \ldots, s_i, t_i, s_{i+1})$ we are done by the inner induction hypothesis and strong determinism of $\mathcal{R}$. So assume $x \in \mathcal{V}(t_{i+1})$. From strong determinism of $\mathcal{R}$, (1), (2), and the inner induction hypothesis we have that $y\sigma'_1 = y\sigma'_2$ for all $y \in \mathcal{V}(s_{i+1})$ and thus $s_{i+1}\sigma'_1 = s_{i+1}\sigma'_2$.

With this we can find a join between $t_{i+1}\sigma'_1$ and $t_{i+1}\sigma'_2$ by applying the induction hypothesis twice as shown in the diagram below:

$$\begin{tikzcd}
 & s_{i+1}\pi^{-}\sigma_1 \arrow[dl, "*"'] \arrow[dr, "*"] & & s_{i+1}\sigma_2 \arrow[dl, "*"'] \arrow[dr, "*"] & \\
t_{i+1}\sigma'_1 \arrow[dr, dotted, "*"] & \text{IH} & s_{i+1}\sigma'_1 \arrow[dl, dotted, "*"'] & & t_{i+1}\sigma'_2 \arrow[ddll, dotted, "*"'] \\
 & \cdot \arrow[dr, dotted, "*"] & & \text{IH} & \\
 & & \cdot & &
\end{tikzcd}$$

Since $t_{i+1}$ is strongly irreducible and $\sigma'_1$ and $\sigma'_2$ are normalized, this yields $t_{i+1}\sigma'_1 = t_{i+1}\sigma'_2$ and thus $x\sigma'_1 = x\sigma'_2$.

3. We are left with the case that there is a variable position $q'$ in $\ell_1$ such that $q = q'r'$ for some position $r'$. Let $x$ be the variable $\ell_1|_{q'}$. Then $x\sigma_1|_{r'} = \ell_2\sigma_2$, which implies $x\sigma_1 \to_{\mathcal{R}}^{*} x\sigma_1[r_2\sigma_2]_{r'}$. Now let $\tau$ be the substitution such that $\tau(x) = x\sigma_1[r_2\sigma_2]_{r'}$ and $\tau(y) = \sigma_1(y)$ for all $y \neq x$. Further, let $\tau'$ be a normalized substitution where $\tau'(y)$ is some normal form of $\tau(y)$ (which we know must exist) for all $y$, so $y\tau \to_{\mathcal{R}}^{*} y\tau'$ for all $y$. Moreover, note that

$$y\sigma_1 \xrightarrow[\mathcal{R}]{*} y\tau \text{ for all } y. \tag{3}$$

We have $u' = \ell_1\sigma_1[r_2\sigma_2]_q = \ell_1\sigma_1[x\tau]_{q'} \to_{\mathcal{R}}^{*} \ell_1\tau$, and thus $u' \to_{\mathcal{R}}^{*} \ell_1\tau'$. From (3) we have $r_1\sigma_1 \to_{\mathcal{R}}^{*} r_1\tau$ and thus $t' = r_1\sigma_1 \to_{\mathcal{R}}^{*} r_1\tau'$. Finally, we show that $\ell_1\tau' \to_{\mathcal{R}} r_1\tau'$, concluding the proof of $t' \downarrow_{\mathcal{R}} u'$. To this end, let $s_i \twoheadrightarrow t_i \in c_1$. By (3) and the definition of $\tau'$ we obtain $s_i\sigma_1 \to_{\mathcal{R}}^{*} t_i\sigma_1 \to_{\mathcal{R}}^{*} t_i\tau'$ and $s_i\sigma_1 \to_{\mathcal{R}}^{*} s_i\tau'$. Then $s_i\tau' \downarrow_{\mathcal{R}} t_i\tau'$ by the induction hypothesis and also $s_i\tau' \to_{\mathcal{R}}^{*} t_i\tau'$, since $t_i$ is strongly irreducible. □

## 5.1 Certification

There are some complications for employing Theorem 6 in practice. Quasi-decreasingness, strong irreducibility, and joinability of CCPs are all undecidable in general. For quasi-decreasingness we fall back to the sufficient criterion that a DCTRS is quasi-decreasing if its unraveling is terminating. This result was formalized by Winkler and Thiemann [43] and is already available in IsaFoR. A sufficient condition for strong irreducibility is *absolute irreducibility.*

**Definition 5 (Absolute irreducibility ☑, absolute determinism ☑)** We call a term $t$ *absolutely $\mathcal{R}$-irreducible* if none of its non-variable subterms unify with any variable-disjoint variant of left-hand sides of rules in the CTRS $\mathcal{R}$. A DCTRS is called *absolutely deterministic* (or ADCTRS for short) if for each rule all right-hand sides of conditions are absolutely $\mathcal{R}$-irreducible.

The proof of the following result [3, Lemma 4.1(a,b)] is immediate.

**Lemma 2** *For a term $t$ and a CTRS $\mathcal{R}$:*
- *If $t$ is absolutely $\mathcal{R}$-irreducible, then $t$ is also strongly $\mathcal{R}$-irreducible.* ☑
- *If $\mathcal{R}$ is absolutely deterministic, then $\mathcal{R}$ is also strongly deterministic.* ☑

We replace joinability of CCPs by infeasibility [30] (already part of IsaFoR) together with two further criteria which rely on *contextual rewriting*.

**Definition 6 (Contextual rewriting ✓)** Consider a set $C$ of equations between terms which we call a *context*.[5] First we define a function $\bar{\cdot}$ on terms such that $\bar{t}$ is the term $t$ where each variable $x \in \mathcal{V}(C)$ is replaced by a fresh constant $\bar{x}$. (Below we sometimes call such constants *Skolem constants*.) Moreover, let $\overline{C}$ denote the set $C$ where all variables have been replaced by fresh constants $\bar{x}$. For a CTRS $\mathcal{R}$ we can make a *contextual rewrite step*, denoted by $s \to_{\mathcal{R},C} t$, if we can make a conditional rewrite step with respect to the CTRS $\mathcal{R} \cup \overline{C}$ from $\bar{s}$ to $\bar{t}$.

We formalize soundness of contextual rewriting [3, Lemma 4.2] as follows:

**Lemma 3** *If $s \to^*_{\mathcal{R},C} t$ then $s\sigma \to^*_{\mathcal{R}} t\sigma$ for all substitutions $\sigma$ satisfying $C$.* ✓

*Proof* Consider the auxiliary function $[t]_\sigma$ that replaces each Skolem constant $\bar{x}$ in $t$ by $\sigma(x)$, that is, it works like applying a substitution to a term, but to Skolem constants instead of variables. Note that $[\bar{t}\,]_\sigma = t\sigma$ whenever $\mathcal{V}(t) \subseteq \mathcal{V}(C)$. Now we show by induction on $n$ that

$$s \to_{\mathcal{R}\cup\overline{C},n} t \text{ implies } [s]_\sigma \to^*_{\mathcal{R},n} [t]_\sigma \qquad (\star)$$

for any $\sigma$ satisfying $C$. The base case is trivial. In the inductive step we have a rule $\ell \to r \Leftarrow c \in \mathcal{R} \cup \overline{C}$, a position $p$, and a substitution $\tau$ such that $s|_p = \ell\tau$, $t = s[r\tau]_p$, and $u\tau \to^*_{\mathcal{R}\cup\overline{C},n} v\tau$ for all $u \approx v \in c$. If $\ell \to r \Leftarrow c$ is a rule in $\mathcal{R}$, then we obtain the contextual rewriting sequence $[u\tau]_\sigma \to^*_{\mathcal{R}\cup\overline{C},n} [v\tau]_\sigma$ for all $u \approx v \in c$ by the induction hypothesis. Then we show $s \to^*_{\mathcal{R}\cup\overline{C},n+1} t$ by induction on the context $s[\cdot]_p$. Otherwise, $\ell \to r \Leftarrow c \in \overline{C}$ and thus $c$ is empty, $\ell\tau = \ell$, and $r\tau = r$, since $\overline{C}$ is an unconditional ground TRS. Moreover, there is a rule $\ell' \to r' \in C$ (thus also $\mathcal{V}(\ell', r') \subseteq \mathcal{V}(C)$) such that $\overline{\ell'} = \ell$ and $\overline{r'} = r$. Again, the final result follows by induction on $s[\cdot]_p$.

Assume $s \to_{\mathcal{R},C} t$. Then $\bar{s} \to_{\mathcal{R}\cup\overline{C},n} \bar{t}$ for some level $n$. Let $\tilde{t}$ denote the extension of $\bar{t}$ where all variables $x$ in $t$ (not just those in $\mathcal{V}(C)$) are replaced by corresponding fresh constants $\bar{x}$. Note that $\tilde{t} = \bar{t}\{x \mapsto \bar{x} \mid x \in \mathcal{V}\}$ for every term $t$. But then we also have $\tilde{s} \to_{\mathcal{R}\cup\overline{C},n} \tilde{t}$ since conditional rewriting is closed under substitutions. Note that $[\tilde{t}\,]_\sigma = t\sigma$ for all $t$. Thus taking $\tilde{s}$ and $\tilde{t}$ for $s$ and $t$ in ($\star$) we obtain $s\sigma \to^*_{\mathcal{R},n} t\sigma$. Since we just established the desired property for single contextual rewrite steps it is straightforward to extend it to rewrite sequences. □

In the following example we illustrate contextual rewriting and how to apply the above lemma.

*Example 9* Consider the CTRS $\mathcal{R}$ consisting of the four rules

$$\mathsf{f}(x,y) \to \mathsf{f}(\mathsf{g}(\mathsf{s}(x)),y) \Leftarrow \mathsf{c}(\mathsf{g}(x)) \twoheadrightarrow \mathsf{c}(\mathsf{a}) \qquad \mathsf{g}(\mathsf{s}(x)) \to x$$
$$\mathsf{f}(x,y) \to \mathsf{f}(x,\mathsf{h}(\mathsf{s}(y))) \Leftarrow \mathsf{c}(\mathsf{h}(y)) \twoheadrightarrow \mathsf{c}(\mathsf{a}) \qquad \mathsf{h}(\mathsf{s}(x)) \to x$$

and the context $C$ containing the two equations

$$\mathsf{c}(\mathsf{g}(x)) \approx \mathsf{c}(\mathsf{a}) \qquad \mathsf{c}(\mathsf{h}(y)) \approx \mathsf{c}(\mathsf{a})$$

[5] This has nothing to do with the usual use of the word *context* in rewriting which refers to a term with a hole.

We have the sequence

$$\mathsf{f}(x,y) \xrightarrow[\mathcal{R},C]{} \mathsf{f}(\mathsf{g}(\mathsf{s}(x)),y) \xrightarrow[\mathcal{R},C]{} \mathsf{f}(\mathsf{g}(\mathsf{s}(x)),\mathsf{h}(\mathsf{s}(y)))$$

The first step is justified by $\mathsf{f}(\bar{\mathsf{x}},\bar{\mathsf{y}}) \to_{\mathcal{R}\cup\overline{C}} \mathsf{f}(\mathsf{g}(\mathsf{s}(\bar{\mathsf{x}})),\bar{\mathsf{y}})$ using the first rule of $\mathcal{R}$ as well as the first equation of $\overline{C}$ to satisfy its condition. For the second step we use $\mathsf{f}(\mathsf{g}(\mathsf{s}(\bar{\mathsf{x}})),\bar{\mathsf{y}}) \to_{\mathcal{R}\cup\overline{C}} \mathsf{f}(\mathsf{g}(\mathsf{s}(\bar{\mathsf{x}})),\mathsf{h}(\mathsf{s}(\bar{\mathsf{y}})))$ employing the second rule of $\mathcal{R}$ and the second equation of $\overline{C}$ to satisfy its condition. From Lemma 3 we get that for all substitutions $\sigma$ that satisfy $C$ we have $\mathsf{f}(x,y)\sigma \to^* \mathsf{f}(\mathsf{g}(\mathsf{s}(x)),\mathsf{h}(\mathsf{s}(y)))\sigma$. In this example the only satisfying substitution is $\sigma = \{x \mapsto \mathsf{s}(\mathsf{a}), y \mapsto \mathsf{s}(\mathsf{a})\}$ employing rules three and four of $\mathcal{R}$.

Lemma 3 is the key to overcome the undecidability issues of conditional rewriting. For example, for joinability of CCPs the problem is that a single joining sequence (as is usual in certificates for TRSs) does not prove joinability for all satisfying substitutions. However, contextual rewriting has this property.

Now we are able to define the two promised criteria for CCPs that employ contextual rewriting: *context-joinability* and *unfeasibility*.

**Definition 7 (Unfeasibility ☑, context-joinability ☑)** Let $s \approx t \Leftarrow c$ be a CCP induced by an overlap between variable-disjoint variants $\ell_1 \to r_1 \Leftarrow c_1$ and $\ell_2 \to r_2 \Leftarrow c_2$ of rules in $\mathcal{R}$ with mgu $\mu$. We say that the CCP is *unfeasible* if we can find terms $u$, $v$, and $w$ such that

1. for all $\sigma$ that satisfy $c$ we have $\ell_1\mu\sigma \succ u\sigma$,
2. $u \to^*_{\mathcal{R},c} v$,
3. $u \to^*_{\mathcal{R},c} w$, and
4. $v$ and $w$ are both strongly irreducible and $v \not\sim w$.

Moreover, we call the CCP *context-joinable* if there exists some term $u$ such that $s \to^*_{\mathcal{R},c} u$ and $t \to^*_{\mathcal{R},c} u$.

*Example 10* Consider the CTRS $\mathcal{R}_{\mathsf{last}}$ consisting of the two rules

$$\mathsf{last}(x : y) \to x \Leftarrow y \twoheadrightarrow \mathsf{nil} \qquad\qquad \mathsf{last}(x : y) \to \mathsf{last}(y) \Leftarrow y \twoheadrightarrow z : v$$

$\mathcal{R}_{\mathsf{last}}$ is quasi-decreasing with respect to some well-founded order $\succ$. Moreover, the CTRS has the CCP $x \approx \mathsf{last}(y) \Leftarrow c$ with $c = \{y \twoheadrightarrow \mathsf{nil},\ y \twoheadrightarrow z : v\}$. This CCP is unfeasible because for all satisfying substitutions $\sigma$ we have $\mathsf{last}(x : y)\sigma \succ y\sigma$, $y \to^*_{\mathcal{R}_{\mathsf{last}},c} z : v$, $y \to^*_{\mathcal{R}_{\mathsf{last}},c} \mathsf{nil}$, and both $z : v$ and $\mathsf{nil}$ are strongly irreducible and not unifiable.

Now, look at the arbitrary CCP $x \approx \mathsf{min}(\mathsf{nil}) \Leftarrow c$ with $c = \{\mathsf{min}(\mathsf{nil}) \twoheadrightarrow x\}$. Since $x \to^*_{\mathcal{R},c} x$ and $\mathsf{min}(\mathsf{nil}) \to^*_{\mathcal{R},c} x$ it is context-joinable (regardless of the actual CTRS $\mathcal{R}$).

Due to Lemma 3 above, context-joinability implies joinability of a CCP for arbitrary satisfying substitutions. The rationale for the definition of unfeasibility is a little bit more technical, since it only makes sense inside the proof (by induction) of the theorem below. Basically, unfeasibility is defined in such a way that unfeasible CCPs contradict the confluence of all $\succ$-smaller terms, which we obtain as induction hypothesis.

In the original paper the definition of quasi-reductivity requires its order to be closed under substitutions. This property is used in the proof of [3, Theorem 4.2]. By a small change to the definition of unfeasibility we avoid this requirement for our extension to quasi-decreasingness.

We are finally ready to state a more applicable version of Theorem 6:

**Theorem 7** *Let the ADCTRS $\mathcal{R}$ be quasi-decreasing. Then $\mathcal{R}$ is confluent if all CCPs are context-joinable, unfeasible, or infeasible.* ☑

*Proof* We denote the well-founded order on terms that we get from quasi-decreasingness by $\succ$. Unfortunately, we cannot directly reuse Theorem 6 and its proof, since we need our sufficient criteria in the induction hypothesis. However, the new proof is quite similar. It only differs in case (1), where we consider a CCP.

a. If the CCP is context-joinable, we obtain a join with respect to contextual rewriting which we can easily transform into a join with respect to $\mathcal{R}$ by an application of Lemma 3 because we have a substitution satisfying the conditions of the CCP.
b. If the CCP is unfeasible, we obtain two diverging contextual rewrite sequences. Again since there is a substitution satisfying the conditions of the CCP we may employ Lemma 3 to get two diverging conditional $\mathcal{R}$-rewrite sequences. Because $\ell_1\sigma \succ_{\mathsf{st}} t_0$ we can use the induction hypothesis to get a join between the two end terms. But from the definition of unfeasibility we also know that the end points are not unifiable (and hence are not the same) and cannot be rewritten (because of strong irreducibility), leading to a contradiction.
c. Finally, if the CCP is infeasible, then there is no substitution that satisfies its conditions, contradicting the fact that we already have such a substitution. □

Have a look at the following example to see Theorem 7 in action.

*Example 11* Consider the quasi-decreasing ADCTRS $\mathcal{R}$ consisting of the following six rules:

$$\mathsf{min}(x : \mathsf{nil}) \to x \qquad (4) \qquad\qquad x < 0 \to \mathsf{false} \qquad (7)$$

$$\mathsf{min}(x : xs) \to x \Leftarrow x < \mathsf{min}(xs) \twoheadrightarrow \mathsf{true} \qquad (5) \qquad\qquad 0 < \mathsf{s}(y) \to \mathsf{true} \qquad (8)$$

$$\mathsf{min}(x : xs) \to \mathsf{min}(xs) \Leftarrow x < \mathsf{min}(xs) \twoheadrightarrow \mathsf{false} \qquad (6) \qquad\qquad \mathsf{s}(x) < \mathsf{s}(y) \to x < y \qquad (9)$$

$\mathcal{R}$ has 6 CCPs, 3 modulo symmetry:

$$x \approx x \Leftarrow x\mathsf{min}(\mathsf{nil}) \twoheadrightarrow \mathsf{true} \qquad (1,2)$$

$$x \approx \mathsf{min}(\mathsf{nil}) \Leftarrow x\mathsf{min}(\mathsf{nil}) \twoheadrightarrow \mathsf{false} \qquad (1,3)$$

$$x \approx \mathsf{min}(xs) \Leftarrow x\mathsf{min}(xs) \twoheadrightarrow \mathsf{true},\ x < \mathsf{min}(xs) \twoheadrightarrow \mathsf{false} \qquad (2,3)$$

To conclude confluence of the system it remains to check its CCPs. The first one, (1,2), is trivially context-joinable because the left- and right-hand sides coincide, (1,3) is infeasible since $\mathsf{tcap}(x < \mathsf{min}(\mathsf{nil})) = x < \mathsf{min}(\mathsf{nil})$ and $\mathsf{false}$ are not unifiable, and (2,3) is unfeasible because with contextual rewriting we can reach the two non-unifiable normal forms $\mathsf{true}$ and $\mathsf{false}$ starting from $x < \mathsf{min}(xs)$. Hence, we conclude confluence of $\mathcal{R}$ by Theorem 7.

## 5.2 Certification Challenges

One of the main challenges towards actual certification is typically disregarded on paper: the definition of critical pairs may yield an infinite set of CCPs even for finite CTRSs. This is because we have to consider arbitrary variable-disjoint variants of rules. However, a hypothetical certificate would only contain those CCPs that were obtained from some specific variable-disjoint variants of rules. Now the argument typically goes as follows: *modulo variable renaming there are only finitely many CCPs. Done.*

However, this reasoning is valid only for properties that are either closed under substitution or at least invariant under renaming of variables. For joinability of plain critical pairs—arguably the most investigated case—this is indeed easy. But when it comes to contextual rewriting we spent a considerable amount of work on some results about permutations that were not available in IsaFoR.

To illustrate the issue, consider the abstract specification of the check function *check-CCPs*, such that *isOK* (*check-CCPs* $\mathcal{R}$) implies that each of the CCPs of $\mathcal{R}$ is either unfeasible, context-joinable, or infeasible. To this end we work modulo the assumption that we already have sound check functions for the latter three properties, which is nicely supported by Isabelle's locale mechanism:

```
locale a194-spec =
  fixes $v_x$ and $v_y$
    and check-context-joinable
    and check-infeasible
    and check-unfeasible
  assumes $v_x$ and $v_y$ are injective
    and $\mathrm{ran}(v_x) \cap \mathrm{ran}(v_y) = \varnothing$
    and isOK (check-context-joinable $\mathcal{R}$ $s$ $t$ $C$) $\Longrightarrow \exists u.\, s \to^*_{\mathcal{R},C} u \land t \to^*_{\mathcal{R},C} u$)
      ...
```

For technical reasons, our formalization uses two locales (`a194-ops`, `a194-spec`) here. We just list the required properties of the renaming functions $v_x$ and $v_y$ and the soundness assumption for *check-context-joinable*.

Now what would a certificate contain and how would we have to check it? Amongst other things, the certificate would contain a finite set of CCPs $\mathcal{C}'$ that were computed by some automated tool. Internally, our certifier computes its own finite set of CCPs $\mathcal{C}$ where variable-disjoint variants of rules are created by fixed injective variable renaming functions $v_x$ and $v_y$, whose ranges are guaranteed to be disjoint. The former prefixes the character "x" and the latter the character "y" to all variable names, hence the names. At this point we have to check that for each CCP in $\mathcal{C}$ there is one in $\mathcal{C}'$ that is its variant, which is not too difficult. More importantly, we have to prove that whenever some desired property $P$, say context-joinability, holds for any CCP, then $P$ also holds for all of its variants (including the one that is part of $\mathcal{C}$).

To this end, assume that we have a CCP resulting from a critical overlap of the two rules $\ell_1 \to r_1 \Leftarrow c_1$ and $\ell_2 \to r_2 \Leftarrow c_2$ at position $p$ with mgu $\mu$. This means that there exist permutations $\pi_1$ and $\pi_2$ such that $(\ell_1 \to r_1 \Leftarrow c_1)\pi_1$ and $(\ell_2 \to r_2 \Leftarrow c_2)\pi_2$ are both in $\mathcal{R}$. In our certifier, mgus are computed by the function $\mathrm{mgu}(s,t)$ which either results in `None`, if $s \not\sim t$, or in `Some` $\mu$ such that $\mu$ is an mgu of $s$ and $t$, otherwise. Moreover, variable-disjointness of rules is ensured by

$v_x$ and $v_y$, so that we actually call $\mathrm{mgu}(\ell_1|_p\pi_1 v_x, \ell_2\pi_2 v_x)$ for computing a concrete CCP corresponding to the one we assumed above. Thus, we need to show that $\mathrm{mgu}(\ell_1|_p, \ell_2) = \texttt{Some}\ \mu$ also implies that $\mathrm{mgu}(\ell_1|_p\pi_1 v_x, \ell_2\pi_2 y_v) = \texttt{Some}\ \mu'$ for some mgu $\mu'$. Moreover, we are interested in the relationship between $\mu$ and $\mu'$ with respect to the variables in both rules. Previously—for an earlier formalization of infeasibility [29]—IsaFoR only contained a result that related both unifiers modulo some arbitrary substitution (that is, not necessarily a renaming).

Unfortunately, contextual rewriting is not closed under arbitrary substitutions. Nevertheless, contextual rewriting is closed under permutations, provided the permutation is also applied to $C$.

**Lemma 4** *For every permutation $\pi$ we have that $s\pi \to^*_{R,C\pi} t\pi$ iff $s \to^*_{R,C} t$.* ☑

It remains to show that $\mu$ and $\mu'$ differ basically only by a renaming (at least on the variables of our two rules), which is covered by the following lemma.

**Lemma 5** *Let $\mathrm{mgu}(s,t) = \texttt{Some}\ \mu$ and $\mathcal{V}(s,t) \subseteq S \cup T$ for two finite sets of variables $S$ and $T$ with $S \cap T = \varnothing$. Then, there exist a substitution $\mu'$ and a permutation $\pi$ such that for arbitrary permutations $\pi_1$ and $\pi_2$: $\mathrm{mgu}(s\pi_1 v_x, t\pi_2 v_y) = \texttt{Some}\ \mu'$, $\mu = \pi_1\mu' v_x\pi\ [S]$, and $\mu = \pi_2\mu' v_y\pi\ [T]$.* ☑

*Proof* Let $h(x) = xv_x\pi_1$ if $x \in S$ and $h(x) = xv_y\pi_2$, otherwise. Then, since $h$ is bijective between $S \cup T$ and $h(S \cup T)$ we can obtain a permutation $\pi$ for which $\pi = h\ [S \cup T]$. We define $\mu' = \pi^-\mu$ and abbreviate $s\pi_1 v_x$ and $t\pi_2 v_y$ to $s'$ and $t'$, respectively. Note that $s' = s\pi$ and $t' = t\pi$. Since $\mu$ is an mgu of $s$ and $t$ we have $s\mu = t\mu$, which further implies $s'\mu' = t'\mu'$. But then $\mu'$ is a unifier of $s'$ and $t'$ and thus there exists some $\mu''$ for which $\mathrm{mgu}(s',t') = \texttt{Some}\ \mu''$ and $s'\mu'' = t'\mu''$.

We now show that $\mu'$ is also most general. Assume $s'\tau = t'\tau$ for some $\tau$. Then $s\pi\tau = t\pi\tau$ and thus there exists some $\delta$ such that $\pi\tau = \mu\delta$ (since $\mu$ is most general). But then $\pi^-\pi\tau = \pi^-\mu\delta$ and thus $\tau = \mu'\delta$. Hence, $\mu'$ is most general.

Since $\mu''$ is most general too, it only differs by a renaming, say $\pi'$, from $\mu'$, that is, $\mu'' = \pi'\mu'$. This yields $\mu = \pi_1\mu'' v_x\pi'^-\ [S]$ and $\mu = \pi_2\mu'' v_y\pi'^-[T]$, and thus concludes the proof. □

## 5.3 Check Functions

Before we can actually certify the output of CTRS confluence tools with CeTA, we have to provide an executable check function for each property that is required to apply Theorem 7 and prove its soundness. For the check functions for infeasibility see Section 7. The check functions for quasi-decreasigness are not described in this article (see [43]). It remains to provide new check functions for absolute irreducibility, absolute determinism, contextual rewrite sequences, context-joinability, and unfeasibility together with their corresponding soundness proofs. For absolute irreducibility we provide the check function *check-airr*, employing existing machinery from IsaFoR for renaming and unification, and prove:

**Lemma 6** *If* $\texttt{isOK}$ *(check-airr $\mathcal{R}$ $t$), then the term $t$ is absolutely $\mathcal{R}$-irreducible.* ☑

This, in turn, is used to define the check function *check-adtrs* and the accompanying lemma for ADCTRSs.

**Lemma 7** *`isOK` (check-adtrs $\mathcal{R}$) iff $\mathcal{R}$ is an ADCTRS.*

Concerning contextual rewriting, we provide the check function *check-csteps* for conditional rewrite sequences together with the following lemma:

**Lemma 8** *Given a CTRS $\mathcal{R}$, a set of conditions $C$, two terms $s$ and $t$, and a list of conditional rewrite proofs ps, we have that `isOK` (check-csteps $(\mathcal{R} \cup \overline{C})$ $\overline{s}$ $\overline{t}$ $\overline{ps}$) implies $s \rightarrow^*_{\mathcal{R},C} t$.*

Although conditional rewriting is decidable in our setting (strong determinism and quasi-decreasingness), we require a *conditional rewrite proof* to provide all the necessary information for checking a single conditional rewrite step (the employed rule, position, and substitution; source and target terms; and recursively, a list of rewrite proofs for each condition of the applied rule). That way, we avoid having to formalize a rewriting engine for conditional rewriting in IsaFoR. With a check function for contextual rewrite sequences in place, we can easily give the check function *check-context-joinable* with the corresponding lemma:

**Lemma 9** *Given a CTRS $\mathcal{R}$, three terms $s$, $t$, and $u$, conditions $C$, and two lists of conditional rewrite proofs ps and qs, we have that*

$$\texttt{isOK}\ (\textit{check-context-joinable}\ (u, ps, qs)\ \mathcal{R}\ s\ t\ C)$$

*implies that there exists some term $u'$ such that $s \rightarrow^*_{\mathcal{R},C} u' \;{}_{\mathcal{R},C}{\overset{*}{\leftarrow}}\; t$.*

Here *check-context-joinable* is a concrete implementation of the homonymous function from the `al94-spec` locale. We further give the check function *check-unfeasible* and the accompanying soundness lemma:

**Lemma 10** *Given a quasi-decreasing CTRS $\mathcal{R}$, two variable-disjoint variants of rules $\rho_1\colon \ell_1 \rightarrow r_1 \Leftarrow c_1$ and $\rho_2\colon \ell_2 \rightarrow r_2 \Leftarrow c_2$ in $\mathcal{R}$, an mgu $\mu$ of $\ell_1|_p$ and $\ell_2$ for some position $p$, a set of conditions $C$ such that $C = c_1\mu, c_2\mu$, three terms $t$, $u$, and $v$, and two lists of conditional rewrite proofs ps and qs, we have that*

$$\texttt{isOK}\ (\textit{check-unfeasible}\ (t, u, v, ps, qs, \rho_1, \rho_2)\ \mathcal{R}\ \ell_1\ \mu\ C)$$

*implies that there exist three terms $t'$, $u'$, and $v'$ such that for all $\sigma$ we have $\ell_1\mu\sigma \succ t'\sigma$, whenever $\sigma$ satisfies $C$, $u' \;{}_{\mathcal{R},C}{\overset{*}{\leftarrow}}\; t' \rightarrow^*_{\mathcal{R},C} v'$, $u'$ and $v'$ are both strongly irreducible, and $u' \not\approx v'$.*

Again, *check-unfeasible* is a concrete implementation of the function of the same name from the `al94-spec` locale and it additionally performs various sanity checks.

At this point, interpreting the `al94-spec` locale using the three check functions *check-context-joinable*, *check-infeasible*, and *check-unfeasible* from above yields the concrete function *check-CCPs*, which is used in the final check *check-al94*.

**Lemma 11** *Given a quasi-decreasing CTRS $\mathcal{R}$, a list of context-joinability certificates c, a list of infeasibility certificates i, and a list of unfeasibility certificates u. Then, `isOK` (check-al94 c i u $\mathcal{R}$) implies confluence of $\mathcal{R}$.*

The formalization of the methods described in this section can be found in the following IsaFoR theory files:

```
thys/Conditional_Rewriting/      thys/Rewriting/
   AL94.thy                         Renaming_Interpretations.thy
   AL94_Impl.thy
   Quasi_Decreasingness.thy
```

In the next section we turn our attention to non-confluence. More specifically, we present simple methods for finding witnesses that establish non-confluence.

## 6 Finding Witnesses for Non-Confluence of CTRSs

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

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

**Lemma 12** *Given a 4-CTRS $\mathcal{R}$ and an unconditional rule $\rho\colon \ell \to r$ in $\mathcal{R}$ where $\mathcal{V}(r) \nsubseteq \mathcal{V}(\ell)$ and $r$ is a normal form with respect to $\mathcal{R}_{\mathsf{u}}$ then $\mathcal{R}$ is non-confluent.* ☑

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

The following example features Cops #320 [27, Example 4.18]:

*Example 12 (Cops #320)* Consider the 4-CTRS $\mathcal{R}$ consisting of the two rules

$$\mathsf{e} \to \mathsf{f}(x) \Leftarrow \mathsf{l} \twoheadrightarrow \mathsf{d} \qquad\qquad \mathsf{A} \to \mathsf{h}(x,x)$$

The right-hand side $\mathsf{h}(x,x)$ of the second (unconditional) rule is a normal form with respect to the underlying TRS $\mathcal{R}_{\mathsf{u}}$ and the only variable occurring in it does not appear in its left-hand side $\mathsf{A}$. So by Lemma 12 $\mathcal{R}$ is non-confluent.

A natural candidate for diverging situations are the critical peaks of a CTRS. We base our next criterion on the analysis of *unconditional* critical pairs (CPs) of CTRSs. This restriction is necessary to guarantee the existence of the actual peak. If we would also allow conditional CPs, we first would have to check for infeasibility, since infeasibility is undecidable in general these checks are potentially very costly (see for example [37]).

**Lemma 13** *Given a CTRS $\mathcal{R}$ and an unconditional CP $s \approx t$ of it. If $s$ and $t$ are not joinable with respect to $\mathcal{R}_{\mathsf{u}}$ then $\mathcal{R}$ is non-confluent.* ☑

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

The following example features Cops #271 [5]:

*Example 13 (Cops #271)* Consider the 3-CTRS $\mathcal{R}$ consisting of the four rules

$$\mathsf{p}(\mathsf{q}(x)) \to \mathsf{p}(\mathsf{r}(x)) \qquad \mathsf{q}(\mathsf{h}(x)) \to \mathsf{r}(x) \qquad \mathsf{r}(x) \to \mathsf{r}(\mathsf{h}(x)) \Leftarrow \mathsf{s}(x) \twoheadrightarrow \mathsf{0} \qquad \mathsf{s}(x) \to \mathsf{1}$$

First of all we can immediately drop the third rule because we can never satisfy its condition and so it does not influence the rewrite relation of $\mathcal{R}$. This results in the TRS $\mathcal{R}'$. Now the left- and right-hand sides of the unconditional CP $\mathsf{p}(\mathsf{r}(z)) \approx \mathsf{p}(\mathsf{r}(\mathsf{h}(z)))$ are not joinable because they are two different normal forms with respect to the underlying TRS $\mathcal{R}'_{\mathsf{u}}$. Hence $\mathcal{R}$ is not confluent by Lemma 13.

While the above lemmas are easy to check and we have fast methods to do so they are also rather ad hoc. A more general but potentially very expensive way to search for non-joinable forks is to use conditional narrowing [22].

**Definition 8 (Conditional narrowing)** Given a CTRS $\mathcal{R}$ we say that $s$ *(conditionally) narrows* to $t$, written $s \rightsquigarrow_\sigma t$ if there is a variant of a rule $\rho\colon \ell \to r \Leftarrow c \in \mathcal{R}$, such that $\mathcal{V}(s) \cap \mathcal{V}(\rho) = \varnothing$ and $u \rightsquigarrow_\sigma^* v$ for all $u \approx v \in c$, a position $p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(s)$, a unifier[6] $\sigma$ of $s|_p$ and $\ell$, and $t = s[r]_p\sigma$. For a narrowing sequence $s_1 \rightsquigarrow_{\sigma_1} s_2 \rightsquigarrow_{\sigma_2} \cdots \rightsquigarrow_{\sigma_{n-1}} s_n$ of length $n$ we write $s_1 \rightsquigarrow_\sigma^n s_n$ where $\sigma = \sigma_1\sigma_2\cdots\sigma_{n-1}$. If we are not interested in the length we also write $s \rightsquigarrow_\sigma^* t$.

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

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

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

**Lemma 14** *Given a CTRS $\mathcal{R}$, if we can find two narrowing sequences $u \rightsquigarrow_\sigma^* s$ and $v \rightsquigarrow_\tau^* t$ such that $u\sigma\mu = v\tau\mu$ for some mgu $\mu$ and $s\sigma\mu$ and $t\tau\mu$ are not $\mathcal{R}_{\mathsf{u}}$-joinable then $\mathcal{R}$ is non-confluent.*

*Proof* Employing Property 1 we immediately get the two rewriting sequences $u\sigma \to_{\mathcal{R}}^* s\sigma$ and $v\tau \to_{\mathcal{R}}^* t\tau$. Since rewriting is closed under substitutions we have $s\sigma\mu \;{}_{\mathcal{R}}^{*}\!\!\leftarrow u\sigma\mu = v\tau\mu \to_{\mathcal{R}}^* t\tau\mu$. As the two endpoints of these forking sequences $s\sigma\mu$ and $t\tau\mu$ are not joinable by $\mathcal{R}_{\mathsf{u}}$ they are certainly also not joinable by $\mathcal{R}$. This establishes non-confluence of the CTRS $\mathcal{R}$. □

*Example 14* Remember the 3-CTRS from Example 7 consisting of the three rules

$$\mathsf{0} + y \to y \qquad \mathsf{s}(x) + y \to x + \mathsf{s}(y) \qquad \mathsf{f}(x, y) \to z \Leftarrow x + y \twoheadrightarrow z + v$$

Starting from a variant of the left-hand side of the third rule $u = \mathsf{f}(x', y')$ we have a narrowing sequence $\mathsf{f}(x', y') \rightsquigarrow_\sigma x_1$ using the variant $\mathsf{f}(x_1, x_2) \to x_3 \Leftarrow x_1 + x_2 \twoheadrightarrow x_3 + x_4$ of the third rule and the substitution $\sigma = \{x' \mapsto x_1, x_3 \mapsto x_1, x_4 \mapsto x_2\}$. We also have another narrowing sequence $\mathsf{f}(x', y') \rightsquigarrow_\tau x_3$ using the same variant of rule three and substitution $\tau = \{x \mapsto x_3 + x_4, x' \mapsto \mathsf{0}, y' \mapsto x_3 + x_4, x_1 \mapsto \mathsf{0}, x_2 \mapsto x_3 + x_4\}$ where for the condition $x_1 + x_2 \twoheadrightarrow x_3 + x_4$ we have the narrowing sequence

6 In our implementation we start from an mgu of $s|_p$ and $\ell$ and extend it while trying to satisfy the conditions.

$x_1 + x_2 \rightsquigarrow_\tau x_3 + x_4$, using a variant of the first rule $\mathsf{0} + x \to x$. Finally, there is an mgu $\mu = \{x_1 \mapsto \mathsf{0}, x_2 \mapsto x_3 + x_4\}$ such that $u\sigma\mu = \mathsf{f}(\mathsf{0}, x_3 + x_4) = u\tau\mu$. Moreover, $x_1\sigma\mu = \mathsf{0}$ and $x_3\tau\mu = x_3$ are two different normal forms. Hence $\mathcal{R}$ is non-confluent by Lemma 14.

## 6.1 Implementation

Starting from its first participation in the confluence competition (CoCo)[7] in 2014 ConCon 1.2.0.3 came equipped with some non-confluence heuristics. Back then it only used Lemmas 12 and 13 and had no support for certification of the output. In the next two years (ConCon 1.3.0 and 1.3.2) we focused on other developments [29, 30, 37, 38] and nothing changed for the non-confluence part. For CoCo 2017 we have added Lemma 14 employing conditional narrowing to ConCon 1.5.1 and the output of all of the non-confluence methods is now certifiable by CeTA.

Our implementation of Lemma 12 takes an unconditional rule $\rho\colon \ell \to r$, a substitution $\sigma = \{x \mapsto y\}$ with $x \in \mathcal{V}(r) \setminus \mathcal{V}(\ell)$ and $y$ fresh with respect to $\rho$ and builds the non-joinable fork $r \;{}_{\rho}\!\leftarrow \ell \to_\rho r\sigma$.

For Lemma 13 we have three concrete implementations that consider an overlap from which an unconditional CP $s \approx t$ arises: The first of which just takes this overlap and then checks that $s$ and $t$ are two different normal forms with respect to $\mathcal{R}_\mathsf{u}$. The second employs the `tcap`-function to check for non-joinability, that is, it checks whether $\mathsf{tcap}(s)$ and $\mathsf{tcap}(t)$ are not unifiable. The third makes a special call to the TRS confluence tool CSI [44] providing the underlying TRS $\mathcal{R}_\mathsf{u}$ as well as the unconditional CP $s \approx t$ where all variables in $s$ and $t$ have been replaced by fresh constants. We issue the following command:

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

The strategy '`(nonconfluence -nonjoinability -steps 0 -tree)[30]`' tells CSI to check non-joinability of two terms using tree automata techniques. Here '`-steps 0`' means that CSI does not rewrite the input terms further before checking non-joinability (this would be unsound in our setting). The timeout is set to 30 seconds. To encode the two terms for which we want to check non-joinability in the input we set CSI to read relative-rewriting input ('`-C RT`'). We provide $\mathcal{R}_\mathsf{u}$ in the usual Cops-format and add one line for the CP $s \approx t$ where its "grounded" left- and right-hand sides are related by '`->=`', that is, we encode it as a relative-rule. This is necessary to distinguish the unconditional CP from the rewrite rules.

Now, for an implementation of Lemma 14 we have to be careful to respect the freshness requirement of the variables in the used rule for every narrowing step with respect to all the previous terms and rules. The crucial point is to efficiently find the two narrowing sequences, to this end we first restrict the set of terms from which to start narrowing. As a heuristic we only consider the left-hand sides of rules of the CTRS under consideration. Next we also prune the search space for narrowing. Here we restrict the length of the narrowing sequences to at most three. In experiments on Cops allowing sequences of length four or more did not yield additional non-confluence proofs but slowed down the tool significantly to the point where we lost other proofs. Further, we also limit the

[7] http://project-coco.uibk.ac.at

recursion depth of conditional narrowing by restricting the level (see the definition of the conditional rewrite relation in the Preliminaries) to at most two. Again, we set this limit as tradeoff after thorough experiments on Cops. Finally, we use Property 1 to translate the forking narrowing sequences into forking conditional rewriting sequences. In this way we generate a lot of forking sequences so we only use fast methods, like non-unifiability of the tcaps of the endpoints or that they are different normal forms, to check for non-joinability of the endpoints. Calls to CSI are to expensive in this context.

## 6.2 Certification

Certification is quite similar for all of the described methods. We have to provide a non-confluence witness, that is, a non-joinable fork. So besides the CTRS $\mathcal{R}$ under investigation we also need to provide the starting term $s$, the two endpoints of the fork $t$ and $u$, as well as certificates for $s \to_{\mathcal{R}}^{+} t$ and $s \to_{\mathcal{R}}^{+} u$, and a certificate that $t$ and $u$ are not joinable. For the forking rewrite sequences we reuse one of our recent formalization [31] to build certificates. We also want to stress that because of Property 1 we did not have to formalize conditional narrowing because going from narrowing to rewrite sequences is already done in ConCon and in the certificate only the rewrite sequences show up. For the non-joinability certificate of $t$ and $u$ there are three options: either we state that $t$ and $u$ are two different normal forms or that $\mathsf{tcap}(t)$ and $\mathsf{tcap}(u)$ are not unifiable; both of these checks are performed within CeTA; or, when the witness was found by an external call to CSI, we just include the generated non-joinability certificate.

The formalization of the methods described in this section can be found in the following IsaFoR theory files:

```
thys/Conditional_Rewriting/Non_Confluence2.thy
```

The next section discusses several methods to show infeasibility of conditional critical pairs. Specifically the techniques described in Sections 4 and 5 benefit from these infeasibility results.

## 7 Infeasibility of Conditional Critical Pairs

The confluence methods detailed in Sections 4 and 5 among other things also analyze the conditional critical pairs of a CTRS. Being able to ignore so called *infeasible* critical pairs simplifies this analysis. See for example Cops #326 [27, $\mathcal{R}_{20}$ in Example 6.7]:

*Example 15 (Cops #326)* The single CCP (modulo symmetry)

$$\mathsf{tp}_2(\mathsf{nil}, x) \approx \mathsf{tp}_2(x : y, z) \Leftarrow \mathsf{isnoc}(\mathsf{nil}) \twoheadrightarrow \mathsf{tp}_2(y, z)$$

of the oriented 3-CTRS $\mathcal{R}$ consisting of the two rules

$$\mathsf{isnoc}(y : \mathsf{nil}) \to \mathsf{tp}_2(\mathsf{nil}, y) \quad \mathsf{isnoc}(x : ys) \to \mathsf{tp}_2(x : xs, y) \Leftarrow \mathsf{isnoc}(ys) \twoheadrightarrow \mathsf{tp}_2(xs, y)$$

is infeasible since $\mathsf{isnoc}(\mathsf{nil})$ is in normal form. Hence $\mathcal{R}$ is orthogonal (modulo infeasibility) and thus confluent.

In this section we present infeasibility methods for oriented 3-CTRSs, one of the most popular types of conditional rewriting. In such systems extra variables in conditions and right-hand sides of rewrite rules are allowed to a certain extend. Moreover, for oriented CTRSs satisfiability of the conditions amounts to reachability. As a consequence of the latter, establishing infeasibility is similar to the problem of eliminating edges in dependency graph approximations, a problem which has been investigated extensively in the literature. The difference is that we deal with CTRSs and the terms we test may share variables.

For brevity, we speak about non-reachability, non-meetability, and non-joinability of two terms $s$ and $t$, when we actually mean that the respective property holds for arbitrary substitution instances $s\sigma$ and $t\tau$.

In the sequel we summarize the methods that we have analyzed and adapted for infeasibility.

## 7.1 Unification

A widely-used method to check for non-reachability is to try to unify the tcap of the source term with the target term; which is the de facto standard for approximating dependency graphs for termination proofs. Remember, the tcap-function approximates the topmost part of a term, its "cap," that does not change under rewriting (see Section 2). It is well known that $\mathsf{tcap}(s) \not\sim t$ implies non-reachability of $t$ from $s$. Typical "pen and paper" definitions (like the one in the preliminaries) rely on replacing subterms by "fresh variables" making them somewhat hard to formalize as already remarked by Thiemann and Sternagel [41]. Instead of inventing fresh variables out of thin air, the IsaFoR-version of tcap replaces every variable occurrence by the symbol $\square$. The resulting terms behave like ground multihole contexts—we call them *ground contexts*—and they are intended to represent the set of all terms resulting from replacing all "holes" by arbitrary terms. This is made formal by the *substitution instance class* of a ground context.

**Definition 9 (Substitution instance class ☑)** The *substitution instance class* $[\![t]\!]$ of a ground context $t$ is defined recursively by.

$$[\![t]\!] := \begin{cases} \mathcal{T}(\mathcal{F},\mathcal{V}) & \text{if } t = \square \\ \{f(s_1,\dots,s_n) \mid s_i \in [\![t_i]\!]\} & \text{if } t = f(t_1,\dots,t_n) \end{cases}$$

Note that for variable-disjoint terms $s$ and $t$, unifiability coincides with $s\sigma = t\tau$ for some, not necessarily identical, substitutions $\sigma$ and $\tau$. Thus asking whether a term $t$ unifies with a variable-disjoint term represented by the ground context $s$ is equivalent to checking whether $t\sigma \in [\![s]\!]$ for some substitution $\sigma$. The latter is called *ground context matching* and shown to be decidable by an efficient algorithm by Thiemann and Sternagel [41]. Thus we can define an efficient executable version of tcap by:

**Definition 10 (Efficient tcap ☑)**

$$\mathsf{tcap}_{\mathcal{R}}(t) = \begin{cases} \square & \text{if } t \text{ is a variable} \\ \square & \text{if } t = f(t_1,\dots,t_n) \text{ and } \ell\sigma \in [\![u]\!] \text{ for some } \sigma \text{ and } \ell \to r \in \mathcal{R} \\ u & \text{otherwise} \end{cases}$$

where $u = f(\mathsf{tcap}_{\mathcal{R}}(t_1), \ldots, \mathsf{tcap}_{\mathcal{R}}(t_n))$ and $\mathcal{R}$ is a TRS. We omit $\mathcal{R}$ if it is clear from context.

This version of tcap is sound, that is, whenever we can reach a term $t$ from an instance of a term $s$ then t is in the substitution instance class of $\mathsf{tcap}(s)$.

**Lemma 15** *If $s\sigma \rightarrow_{\mathcal{R}}^* t$ then $t \in [\![\mathsf{tcap}(s)]\!]$.* ✓

Then checking non-reachability of $t$ from $s$ amounts to deciding whether there does not exist a substitution $\tau$ such that $t\tau \in [\![\mathsf{tcap}(s)]\!]$, for which we use the more succinct notation $\mathsf{tcap}(s) \not\sim t$ almost everywhere else in this article.

While the above definition of tcap and the corresponding soundness lemma were already present in IsaFoR, the following easy extension also allows us to test for non-joinability.

**Lemma 16** *If $s\sigma \rightarrow_{\mathcal{R}}^* \cdot {}_{\mathcal{R}}^*\!\leftarrow t\tau$ then $[\![\mathsf{tcap}(s)]\!] \cap [\![\mathsf{tcap}(t)]\!] \neq \varnothing$.* ✓

*Proof* We have $s\sigma \rightarrow_{\mathcal{R}}^* u$ and $t\tau \rightarrow_{\mathcal{R}}^* u$ for some $u$. By Lemma 15 we have $u \in \mathsf{tcap}(s)$ and $u \in \mathsf{tcap}(t)$. □

Fortunately the same techniques that are used to obtain an algorithm for ground context matching can be reused for *ground context unifiability*, that is, checking $[\![\mathsf{tcap}(s)]\!] \cap [\![\mathsf{tcap}(t)]\!] \neq \varnothing$ (elsewhere in this article we use the notation $\mathsf{tcap}(s) \not\sim \mathsf{tcap}(t)$).

Now for checking infeasibility of a CCP we use the underlying TRS and if there is more than one condition we collect the left- and right-hand sides separately under a fresh function symbol as follows:

**Corollary 2 (Infeasibility via tcap)** *Let $\mathcal{R}$ be an oriented 3-CTRS. A CCP*

$$u \approx v \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k$$

*is infeasible if $\mathsf{tcap}_{\mathcal{R}_\mathsf{u}}(\mathsf{cs}(s_1, \ldots, s_k)) \not\sim \mathsf{cs}(t_1, \ldots, t_k)$ (where cs is a fresh function symbol of arity $k$).* *(Combination of ✓ and ✓)*

*Example 16* Consider Cops #326 from Example 15. The CCP

$$\mathsf{tp}_2(\mathsf{nil}, x) \approx \mathsf{tp}_2(x : y, z) \Leftarrow \mathsf{isnoc}(\mathsf{nil}) \twoheadrightarrow \mathsf{tp}_2(y, z)$$

is infeasible by Corollary 2 because $\mathsf{tcap}_{\mathcal{R}_\mathsf{u}}(\mathsf{isnoc}(\mathsf{nil})) = \mathsf{isnoc}(\mathsf{nil}) \not\sim \mathsf{tp}_2(x_1, x_2) = \mathsf{tcap}_{\mathcal{R}_\mathsf{u}}(\mathsf{tp}_2(y, z))$.

## 7.2 Symbol Transition Graph

One shortcoming of the tcap method is that although later during unification we have to consider the target term anyway it first only considers the starting term. Maybe we could gain power if we use knowledge about the structure of the target from the start? This consideration is the basis for the so called *symbol transition graph* [35]. The idea is simple enough, we consider the root symbols of left- and right-hand sides of rewrite rules of a TRS (for CTRSs we perform an overapproximation by using the underlying TRS)[8] and collect the resulting dependencies in a graph.

[8] There is an inductive version of the symbol transition graph specifically for CTRSs due to Sternagel and Yamada [35]. However, it is not yet formalized and thus we concentrate on the TRS version, which we formalized in Isabelle/HOL.

**Definition 11 (Symbol transition graph)** Given a TRS $\mathcal{R}$ the edges of its *symbol transition graph* are given by the relation

$$\sqsupset_{\mathsf{stg}} := \{(\mathsf{root}(\ell), \mathsf{root}(r)) \mid \ell \to r \in \mathcal{R}\}$$

To clarify this concept let us first look at some examples.

*Example 17* Below we give the symbol transition graphs of the following three TRSs $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$, respectively.

$\mathcal{R}_1$:

$$\mathsf{f}(x, x, x) \to \mathsf{f}(x, \mathsf{0}, \mathsf{1})$$

$$\begin{tikzcd} \mathsf{f} \arrow[loop below] \end{tikzcd}$$

$\mathcal{R}_2$:

$$\mathsf{0} + y \to y$$
$$\mathsf{s}(x) + y \to \mathsf{s}(x + y)$$

$$\begin{tikzcd} & \mathsf{s} \\ + \arrow[ur] \arrow[dr] & \\ & \bot \end{tikzcd}$$

$\mathcal{R}_3$:

$$\mathsf{f}(x, y) \to \mathsf{a}$$
$$\mathsf{h}(x) \to \mathsf{g}(x, x)$$
$$\mathsf{h}(\mathsf{g}(x, y)) \to \mathsf{h}(\mathsf{f}(x, y))$$

$$\begin{tikzcd} \mathsf{f} \arrow[r] & \mathsf{a} \\ \mathsf{h} \arrow[r] \arrow[loop below] & \mathsf{g} \end{tikzcd}$$

*Example 18* For the CTRS of Example 15 the symbol transition graph consists of a single edge:

$$\begin{tikzcd} \mathsf{isnoc} \arrow[r] & \mathsf{tp}_2 \end{tikzcd}$$

Now we can use the following lemma for reachability analysis of TRSs.

**Lemma 17** *For two terms $s = f(\ldots)$ and $t = g(\ldots)$ if $s \to^* t$ then either $f = g$, $f \sqsupset^+_{\mathsf{stg}} g$, or $f \sqsupset^+_{\mathsf{stg}} \bot$.*

*Proof* We prove this by induction on the length of the rewrite sequence $s \to^k t$. In the base case we have $s = t$ and hence $f = g$. Now in the step case we look at the sequence $s \to u \to^k t$. If $u$ is a variable then $s \to u$ is a root step with a collapsing rule and hence $f \sqsupset_{\mathsf{stg}} \bot$ and thus also $f \sqsupset^+_{\mathsf{stg}} \bot$. Otherwise there is some function symbol $h$ such that $u = h(\ldots)$. If we have a non-root step then $f = h$ and we are done. Otherwise we have $f \sqsupset_{\mathsf{stg}} h$. From the induction hypothesis we know that one of $h = g$, $h \sqsupset^+_{\mathsf{stg}} g$, or $h \sqsupset^+_{\mathsf{stg}} \bot$ holds. So either $f \sqsupset_{\mathsf{stg}} h = g$, $f \sqsupset_{\mathsf{stg}} h \sqsupset^+_{\mathsf{stg}} g$, or $f \sqsupset_{\mathsf{stg}} h \sqsupset^+_{\mathsf{stg}} \bot$, which finishes the proof. □

From the previous lemma we immediately get the following non-reachability result:

**Corollary 3** *If $f \neq g$ and neither $f \sqsupset^+_{\mathsf{stg}} g$ nor $f \sqsupset^+_{\mathsf{stg}} \bot$ then $f(\ldots) \not\to^* g(\ldots)$.*

*Example 19* Consider the TRS $\mathcal{R}_3$ from Example 17. Do we have a rewrite sequence $\mathsf{f}(x, y)\sigma \to^* \mathsf{g}(x, y)\tau$ for some substitutions $\sigma$ and $\tau$? Since $\mathsf{tcap}(\mathsf{f}(x, y)) = z \sim \mathsf{g}(x, y)$ we cannot conclude non-reachability using $\mathsf{tcap}$. Then again $\mathsf{f} \neq \mathsf{g}$, $\mathsf{f} \not\sqsupset^+_{\mathsf{stg}} \mathsf{g}$, and $\mathsf{f} \not\sqsupset^+_{\mathsf{stg}} \bot$. So $\mathsf{g}(x, y)$ is not reachable from $\mathsf{f}(x, y)$ with respect to $\mathcal{R}_3$ by Corollary 3.

*Example 20* For Cops #547 (see also Example 4) to get infeasibility of its CCP we have to show that either $x\sigma \not\to^* \mathsf{a}\tau$, $x\sigma \not\to^* \mathsf{b}\tau$, or $\mathsf{cs}(x, x)\sigma \not\to^* \mathsf{cs}(\mathsf{a}, \mathsf{b})\tau$ for a fresh compound symbol $\mathsf{cs}$ and any two substitutions $\sigma$ and $\tau$. Unfortunately, we cannot employ Corollary 3 here. If we look at the two conditions separately the left-hand sides are variables and looking at the combined conditions we have the same function symbol $\mathsf{cs}$ on the left- and right-hand sides.

In the next section we will see a way to combine the best of both tcap and the symbol transition graph.

## 7.3 Decomposing Reachability Problems

Both, the tcap method and the symbol transition graph do have their pros and cons. For example, in contrast to the tcap method, the symbol transition graph takes information about the target term into account, then again, it does not recursively look at the whole term, like tcap, but only at the root symbols.

In this section we introduce a modular framework for reachability, that allows us to decompose a given problem into several smaller ones. The binary relation $\sqsupset$ that we employ to show non-reachability between two terms has to have certain properties to be usable for decomposition. These properties are summarized in the following definition (where $\xrightarrow{>\epsilon}$ denotes rewrite steps below the root position).

**Definition 12 (Decomposition ✓)** A binary relation on terms $\sqsupset$ *admits decomposition* if both

1. $s\sigma \sqsupset t\tau$ implies $s \sqsupset t$ for all substitutions $\sigma$ and $\tau$ and
2. $s \not\sqsupset t$ together with $s \to^* t$ implies $s \xrightarrow{>\epsilon}{}^* t$.

Using the binary relation $\sqsupset$ we can define a *decomposition function* that deconstructs a single reachability problem into several (smaller) ones.

**Definition 13 (Abstract decomposition function ✓)** The *abstract decomposition function* $\mathsf{D}_{s,t}$ takes a reachability problem $(s, t)$ (meaning: “Is $t$ reachable from $s$?”) as input and recursively computes a set of (possibly easier) reachability problems with respect to the relation $\sqsupset$.

$$\mathsf{D}_{s,t} := \begin{cases} \mathsf{D}_{s_1,t_1} \cup \cdots \cup \mathsf{D}_{s_n,t_n} & \text{if } s = f(s_1, \ldots, s_n), t = f(t_1, \ldots, t_n), \text{ and } s \not\sqsupset t \\ \{(s, t)\} & \text{otherwise} \end{cases}$$

With this function in place we can now formally define when decomposition is admissible for a reachability problem.

**Lemma 18** *If $\sqsupset$ admits decomposition and $s\sigma \to^* t\tau$ then also $u\sigma \to^* v\tau$ for all $(u, v) \in \mathsf{D}_{s,t}$.* ✓

Of course in our setting we want to use decomposition of a reachability problem to show non-reachability. To do that we first have to specify an abstract non-reachability test that we later instantiate with concrete checks.

**Definition 14 (Abstract non-reachability check)** If two functional terms $s = f(...)$ and $t = g(...)$ have different root symbols and $s$ and $t$ are not in the relation $\sqsupset$ then $t$ is not reachable from $s$. Otherwise it is. This is tested by the *abstract non-reachability check* $\mathsf{nonreach}(s,t)$ defined as follows:

$$\mathsf{nonreach}(s,t) := \begin{cases} f \neq g \wedge s \not\sqsupset t & \text{if } s = f(...) \text{ and } t = g(...) \\ \mathsf{false} & \text{otherwise} \end{cases}$$

Finally, we are ready to state a non-reachability lemma.

**Lemma 19** *If $\sqsupset$ admits decomposition and* $\mathsf{nonreach}(s,t)$ *holds then $s\sigma \not\to^* t\tau$.*

*Proof* Assume to the contrary that $s\sigma \to^* t\tau$. Because $\sqsupset$ admits decomposition we have

1. $s\sigma \sqsupset t\tau \Longrightarrow s \sqsupset t$
2. $s \not\sqsupset t \Longrightarrow s \to^* t \Longrightarrow s \xrightarrow{>\epsilon}{}^* t$

From 1 and $\mathsf{nonreach}(s,t)$ we have that $s\sigma \not\sqsupset t\tau$. From this, our starting assumption, and 2 we have $s\sigma \xrightarrow{>\epsilon}{}^* t\tau$. But this implies $\mathsf{root}(s) = \mathsf{root}(t)$ which contradicts $\mathsf{nonreach}(s,t)$. □

Below we give two possible instances of $\sqsupset$ one using the $\mathsf{tcap}$-function of Section 7.1 and the other using the symbol transition graph $\sqsupset_{\mathsf{stg}}^{+}$ of Section 7.2.

**Definition 15** **($\sqsupset_{\mathsf{tcap}}$, $\sqsupset_{\mathsf{rs}}$)** We define

– $s \sqsupset_{\mathsf{tcap}} t$ iff there exists a rule $\ell \to r$ in $\mathcal{R}$ such that $\mathsf{tcap}(s) \sim \ell$, and
– $s \sqsupset_{\mathsf{rs}} t$ iff at least one of the following properties holds: $s$ is a variable, $t$ is a variable, $\mathsf{root}(s) \sqsupset_{\mathsf{stg}}^{+} \mathsf{root}(t)$, or $\mathsf{root}(s) \sqsupset_{\mathsf{stg}}^{+} \bot$.

*Example 21* Consider the TRS $\mathcal{R}_4$ consisting of the four rules

$$\mathsf{f}(x) \to \mathsf{a} \qquad \mathsf{p}(\mathsf{s}(x)) \to \mathsf{b} \qquad \mathsf{h}(x) \to \mathsf{g}(x) \qquad \mathsf{h}(\mathsf{f}(\mathsf{b})) \to \mathsf{h}(\mathsf{f}(\mathsf{p}(\mathsf{a})))$$

and the two terms $s = \mathsf{f}(\mathsf{p}(\mathsf{a}))$ and $t = \mathsf{f}(\mathsf{b})$. We have $s \sqsupset_{\mathsf{tcap}} t$ since $\mathsf{tcap}(s) = x_1$ unifies with the left-hand side of every rule in $\mathcal{R}_4$ but $s \not\sqsupset_{\mathsf{rs}} t$ because $s$ and $t$ are neither variables, nor $\mathsf{f} \sqsupset_{\mathsf{stg}}^{+} \mathsf{f}$, nor $\mathsf{f} \sqsupset_{\mathsf{stg}}^{+} \bot$. Moreover, for the two terms $s' = \mathsf{p}(\mathsf{a})$ and $t' = \mathsf{b}$ we have $s' \not\sqsupset_{\mathsf{tcap}} t'$ since $\mathsf{tcap}(s') = \mathsf{p}(\mathsf{a})$ does not unify with the left-hand side of any rule in $\mathcal{R}_4$ but $s' \sqsupset_{\mathsf{rs}} t'$ because the second rule of $\mathcal{R}_4$ yields $\mathsf{p} \sqsupset_{\mathsf{stg}} \mathsf{b}$.

Both of these relations admit decomposition of reachability problems.

**Lemma 20** *Both, $\sqsupset_{\mathsf{tcap}}$ and $\sqsupset_{\mathsf{rs}}$ admit decomposition.*

*Proof* We first consider $\sqsupset_{\mathsf{tcap}}$. Assume $s\sigma \sqsupset_{\mathsf{tcap}} t\tau$, so there exists a rule $\ell \to r \in \mathcal{R}$ such that $\mathsf{tcap}(s\sigma) \sim \ell$. Remember that the latter is just a shorthand for the existence of a substitution $\mu$ such that $\ell\mu \in [\![s\sigma]\!]$. By induction on $u$ we have that $[\![\mathsf{tcap}(u\mu)]\!] \subseteq [\![\mathsf{tcap}(u)]\!]$ for any substitution $\mu$. Hence also $\ell\mu \in [\![s]\!]$ and thus $\mathsf{tcap}(s) \sim \ell$. But then by definition $s \sqsupset_{\mathsf{tcap}} t$. Now assume $s \to^* t$ and $s \not\sqsupset_{\mathsf{tcap}} t$. From the latter we have that $\mathsf{tcap}(s) \not\sim \ell$ for all $\ell \to r \in \mathcal{R}$. If there would be any root step in $s \to^* t$ then $\mathsf{tcap}(s) \sim \ell$ for some rule $\ell \to r \in \mathcal{R}$. Hence $s \xrightarrow{>\epsilon}{}^* t$. So $\sqsupset_{\mathsf{tcap}}$ admits decomposition.

Next consider $\sqsupset_{\mathsf{rs}}$. Assume $s\sigma \sqsupset_{\mathsf{rs}} t\tau$. So by definition either $s\sigma \in \mathcal{V}$, $t\tau \in \mathcal{V}$, $\mathsf{root}(s\sigma) \sqsupset_{\mathsf{stg}}^{+} \mathsf{root}(t\tau)$, or $\mathsf{root}(s\sigma) \sqsupset_{\mathsf{stg}}^{+} \bot$. But then obviously $s \sqsupset_{\mathsf{rs}} t$ because for any term $u$ and any substitution $\mu$ if $u\mu \in \mathcal{V}$ certainly also $u \in \mathcal{V}$ and if $u \notin \mathcal{V}$ then $\mathsf{root}(u\mu) = \mathsf{root}(u)$. Now assume $s \to^* t$ and $s \not\sqsupset_{\mathsf{rs}} t$. From the latter we have that neither $s$ nor $t$ are variables, as well as $\mathsf{root}(s) \not\sqsupset_{\mathsf{stg}}^{+} \mathsf{root}(t)$ and $\mathsf{root}(s) \not\sqsupset_{\mathsf{stg}}^{+} \bot$. But that means that there is no root step in $s \to^* t$ and hence $s \xrightarrow{>\epsilon}{}^* t$. So also $\sqsupset_{\mathsf{rs}}$ admits decomposition. □

Furthermore, if two relations admit decomposition then also their intersection does.

**Lemma 21** *If $\sqsupset_1$ and $\sqsupset_2$ admit decomposition then so does $\sqsupset_1 \cap \sqsupset_2$.* ☑

So in our implementation we employ the intersection of the two relations defined earlier and obtain the following result by combining Lemma 19, Lemma 20, and Lemma 21:

**Corollary 4** *If $\mathsf{nonreach}(s,t)$ holds for $\sqsupset = \sqsupset_{\mathsf{tcap}} \cap \sqsupset_{\mathsf{rs}}$, then $s\sigma \not\to^* t\tau$.*

We call the above instance of $\mathsf{nonreach}$ *generalized* $\mathsf{tcap}$.

This is stronger than relying on the two relations separately as shown in the following example.

*Example 22* Consider the TRS $\mathcal{R}_4$ from Example 21 above. Look at the two terms $s = \mathsf{f}(\mathsf{p}(\mathsf{a}))$ and $t = \mathsf{f}(\mathsf{b})$ and assume we want to know whether $s\sigma \to^* t\tau$ for any substitutions $\sigma$ and $\tau$. We employ the abstract non-reachability check and the abstract decomposition function. In our first attempt we instantiate $\sqsupset$ with $\sqsupset_{\mathsf{rs}}$. Since the root symbols of $s$ and $t$ are the same $\mathsf{nonreach}(s,t)$ does not hold. We try to decompose the problem and get $\mathsf{D}_{s,t} = \{(\mathsf{p}(\mathsf{a}),\mathsf{b})\}$ because $s \not\sqsupset_{\mathsf{rs}} t$. Unfortunately, $\mathsf{p}(\mathsf{a}) \sqsupset_{\mathsf{rs}} \mathsf{b}$ which means that $\mathsf{nonreach}(\mathsf{p}(\mathsf{a}),\mathsf{b})$ is not true. Clearly $\sqsupset_{\mathsf{rs}}$ does not work.

Let us try to instantiate $\sqsupset$ with $\sqsupset_{\mathsf{tcap}}$. Since $s \sqsupset_{\mathsf{tcap}} t$ neither $\mathsf{nonreach}(s,t)$ holds nor can we decompose the problem and we immediately have to give up.

So let's finally try to instantiate $\sqsupset$ with $\sqsupset_{\mathsf{tcap}} \cap \sqsupset_{\mathsf{rs}}$. The root symbols of $s$ and $t$ are the same so $\mathsf{nonreach}(s,t)$ does not hold. Since $s \not\sqsupset_{\mathsf{rs}} t$ also $s\ (\sqsupset_{\mathsf{tcap}} \cap \sqsupset_{\mathsf{rs}})\ t$ does not hold and thus $\mathsf{D}_{s,t} = \{(\mathsf{p}(\mathsf{a}),\mathsf{b})\}$. We have $\mathsf{p}(\mathsf{a}) \not\sqsupset_{\mathsf{tcap}} \mathsf{b}$ and thus $\mathsf{nonreach}(\mathsf{p}(\mathsf{a}),\mathsf{b})$, which together with Lemma 19 yields $\mathsf{p}(\mathsf{a})\sigma \not\to^* \mathsf{b}\tau$. From this we get non-reachability of $t$ from $s$ by Lemma 18 and we are done.

## 7.4 Exact Tree Automata Completion

What is generally known as tree automata completion today was introduced by Genet in 1998 [8, 9]. Already in 1996 Jacquemard [19] used a similar concept to show decidability of reachability for linear and growing TRSs. His proof was based on the construction of a tree automaton that accepts the set of ground terms which are normalizable with respect to a given linear and growing TRS $\mathcal{R}$. If we replace the automaton recognizing $\mathcal{R}$-normal forms in Jacquemard's construction by an arbitrary automaton $\mathcal{A}$ we arrive at a tree automaton that accepts the $\mathcal{R}$-ancestors of the language of $\mathcal{A}$.

We need some basic definitions and auxiliary lemmas before we present the construction of this *ancestor automaton* in detail.

**Definition 16 (Ground-instances )** The set of *ground-instances* of a term $t$, that is, the set of terms $s$ such that $s = t\sigma$ for some ground substitution $\sigma$ is denoted by $\Sigma(t)$.

**Definition 17 (Growingness, linear growing approximation)** A TRS $\mathcal{R}$ is called *growing* if for all $\ell \to r \in \mathcal{R}$ the variables in $\mathcal{V}(\ell) \cap \mathcal{V}(r)$ occur at depth at most one in $\ell$. Given a TRS $\mathcal{R}$ the *linear growing approximation* is defined as any linear growing TRS obtained from $\mathcal{R}$ by linearizing the left-hand sides, renaming the variables in the right-hand sides that occur at a depth greater than one in the corresponding left-hand side, and finally also linearizing the right-hand sides.[9] The linear growing approximation of a TRS $\mathcal{R}$ is denoted by $\mathsf{g}(\mathcal{R})$.

**Definition 18 (Ground-instance transitions $\Delta_t$ )** Let $[t]$ denote a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ where all variable-occurrences have been replaced by a fresh symbol $\square$ (similar to Definitions 9 and 10). Using such terms as states we define the set $\Delta_t$ that contains all transitions which are needed to recognize all ground-instances of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ in state $[t]$.

$$\Delta_t = \begin{cases} \{f([t_1], \ldots, [t_n]) \to [t]\} \cup \bigcup_{1 \leqslant i \leqslant n} \Delta_{t_i} & \text{if } t = f(t_1, \ldots, t_n) \\ \{f(\square, \ldots, \square) \to \square \mid f \in \mathcal{F}\} & \text{otherwise} \end{cases}$$

Note that if $t$ is not linear this actually gives an overapproximation. The next lemma holds by definition of $\Delta_t$.

**Lemma 22** *For any subterm $s$ of any term $t$ if there is a sequence $u \to_{\Delta_t}^{+} [s]$ then $u$ is a ground-instance of $s$, and vice versa if $t$ is linear.*

We now use $\Delta_t$ to define an automaton for the ground-instances of $t$.

**Definition 19 (Ground-instance automaton $\mathcal{A}_{\Sigma(t)}$ )** Let $Q_t$ denote the set of states occurring in $\Delta_t$ then we call the tree automaton $\mathcal{A}_{\Sigma(t)} = \langle \mathcal{F}, Q_t, \{[t]\}, \Delta_t \rangle$ the *ground-instance automaton for $t$*.

**Lemma 23** *The language of $\mathcal{A}_{\Sigma(t)}$ is an overapproximation of the set of ground-instances of $t$ in general and an exact characterization if $t$ is linear.*

Using the concept of ground-instance automaton we are now able to define a tree automaton which accepts all $\mathcal{R}$-ancestors of a given regular set of ground terms using exact tree automata completion.

**Definition 20 (Ancestor automaton $\mathsf{anc}_{\mathcal{R}}(\mathcal{A})$ )** Given a tree automaton $\mathcal{A} = \langle \mathcal{F}, Q_{\mathcal{A}}, Q_f, \Delta \rangle$ whose states are all accessible, and a linear and growing TRS $\mathcal{R}$ the construction proceeds as follows.

First we extend the set of transitions of $\mathcal{A}$ in such a way that we can match left-hand sides of rules in $\mathcal{R}$. This yields the set of transitions $\Delta_0 = \Delta \cup \bigcup_{\ell \to r \in \mathcal{R}} \Delta_\ell$. Let $\mathcal{A}_0 = \langle \mathcal{F}, Q, Q_f, \Delta_0 \rangle$ where $Q$ denotes the set of states in $\Delta_0$. We have to ensure (for example by using the disjoint union of states) that for any state $q$ which is used in both $\Delta$ and some $\Delta_\ell$, the terms which can reach it are the same

[9] Note that this definition of the linear growing approximation is ambiguous, because the second step depends on the first step and we have a choice of how to linearize the variables (see also [21]).

($\{t \mid t \rightarrow^+_{\Delta} q\} = \{t \mid t \rightarrow^+_{\Delta_\ell} q\}$). Then the language does not change, that is, $L(\mathcal{A}_0) = L(\mathcal{A})$.

Finally, we saturate $\Delta_0$ by inference rule (†) in order to extend the language by $\mathcal{R}$-ancestors, that is, if we can reach a state $q$ from an instance of a right-hand side of a rule in $\mathcal{R}$ we add a transition which ensures that $q$ is reachable from the corresponding left-hand side:[10]

$$\frac{f(\ell_1,\ldots,\ell_n) \to r \in \mathcal{R} \quad r\theta \rightarrow^*_{\Delta_k} q}{f(q_1,\ldots,q_n) \to q \in \Delta_{k+1}} \tag{†}$$

Here $\theta\colon \mathcal{V}(r) \to Q$ is a state substitution and $q_i = \ell_i\theta$ if $\ell_i$ is a variable in $r$ and $q_i = [\ell_i]$ otherwise. Note that this inductive definition possibly adds many new transitions from $\Delta_k$ to $\Delta_{k+1}$.

Since $\mathcal{R}$ is finite, the number of states is finite, and we do not introduce new states using (†), this process terminates after finitely many steps resulting in the set of transitions $\Delta_m$. Also note that $\Delta_k$ is monotone with respect to $k$, that is, $\Delta_k \subseteq \Delta_{k+1}$ for all $k \geqslant 0$. We call $\mathsf{anc}_{\mathcal{R}}(\mathcal{A}) = \langle \mathcal{F}, Q, Q_f, \Delta_m\rangle$ the $\mathcal{R}$-ancestors automaton for $\mathcal{A}$. It is easy to show that $L(\mathcal{A}_0) \subseteq L(\mathsf{anc}_{\mathcal{R}}(\mathcal{A}))$.

**Theorem 8** *Given a tree automaton $\mathcal{A}$ as well as a linear and growing TRS $\mathcal{R}$ the language of $\mathsf{anc}_{\mathcal{R}}(\mathcal{A})$ is exactly the set of $\mathcal{R}$-ancestors of $L(\mathcal{A})$.* ☑

*Proof* First we prove that $(\rightarrow^*_{\mathcal{R}})[L(\mathcal{A})] \subseteq L(\mathsf{anc}_{\mathcal{R}}(\mathcal{A}))$. Pick a term $s \in (\rightarrow^*_{\mathcal{R}})[L(\mathcal{A})]$. That means that there is a rewrite sequence $s \rightarrow^k_{\mathcal{R}} t$ of length $k \geqslant 0$ for some $t \in L(\mathcal{A})$. We proceed by induction on $k$. If $k = 0$ then $s = t$ and hence $s \in L(\mathsf{anc}_{\mathcal{R}}(\mathcal{A}))$. Now assume $k = k' + 1$ for some $k' \geqslant 0$ then there is a rewrite sequence $s = C[f(\ell_1,\ldots,\ell_n)\sigma] \rightarrow_{\mathcal{R}} C[r\sigma] \rightarrow^{k'}_{\mathcal{R}} t$ for some context $C$, rewrite rule $f(\ell_1,\ldots,\ell_n) \to r \in \mathcal{R}$, and substitution $\sigma$. By induction hypothesis $C[r\sigma] \in L(\mathsf{anc}_{\mathcal{R}}(\mathcal{A}))$. But that means that there is a state substitution $\theta\colon \mathcal{V}(r) \to Q$, a state $q \in Q$, and a final state $q_f \in Q_f$ such that $C[r\sigma] \rightarrow^*_{\Delta_m} C[r\theta] \rightarrow^*_{\Delta_m} C[q] \rightarrow^*_{\Delta_m} q_f$. From the construction using rule (†) we know that there is a transition $f(q_1,\ldots,q_n) \to q \in \Delta_m$ such that $q_i = \ell_i\theta$ if $\ell_i \in \mathcal{V}(r)$ and $q_i = [\ell_i]$ otherwise. If $\ell_i \in \mathcal{V}(r)$ then $\ell_i\sigma \rightarrow^+_{\Delta_m} \ell_i\theta$ and otherwise $\ell_i\sigma \rightarrow^+_{\Delta_m} [\ell_i]$ for all $1 \leqslant i \leqslant n$. Hence in both cases $\ell_i\sigma \rightarrow^+_{\Delta_m} q_i$. But then we can construct the sequence $s = C[f(\ell_1\sigma,\ldots,\ell_n\sigma)] \rightarrow^*_{\Delta_m} C[f(q_1,\ldots,q_n)] \rightarrow_{\Delta_m} C[q] \rightarrow^*_{\Delta_m} q_f$ and hence $s \in L(\mathsf{anc}_{\mathcal{R}}(\mathcal{A}))$.

For the other direction we prove the following two properties for all sequences $s \rightarrow^+_{\Delta_m} q$:

1. If $q = [t]$ for some subterm of a left-hand side of a rule in $\mathcal{R}$ then $s \in (\rightarrow^*_{\mathcal{R}})[\Sigma(t)]$.
2. If $q \in Q_f$ then $s \in (\rightarrow^*_{\mathcal{R}})[L(\mathcal{A})]$.

The proof for both properties works along the same lines. We sketch the one for the first property here. From the construction using rule (†) we know that $s \rightarrow^+_{\Delta_k} [t]$ for some $k \geqslant 0$. We proceed by induction on $k$. If $k = 0$ then $s \rightarrow^+_{\Delta_0} [t]$. By construction of $\mathcal{A}_0$ and Lemma 22 we have $s \in \Sigma(t)$ and hence also $s \in (\rightarrow^*_{\mathcal{R}})[\Sigma(t)]$. Now assume that $k = k' + 1$ for some $k' \geqslant 0$. By induction hypothesis $s \rightarrow^+_{\Delta_{k'}} [t]$ implies $s \in (\rightarrow^*_{\mathcal{R}})[\Sigma(t)]$ for all terms $s$ and $t$. Consider the set $\Delta_{k'+1} \setminus \Delta_{k'}$ of transitions which were newly added in $\Delta_{k'+1}$. We continue by a second induction

[10] This is symmetric to resolving compatibility violations in the tree automata completion by Genet [8, 9].

$$\begin{tikzcd}
s = D[f(s_1,\ldots,s_n)] \arrow[r, "*", "\Delta\cup\Delta_{k'}"'] \arrow[d, dotted, "\mathcal{R}"', "*"] & D[f(q_1,\ldots,q_n)] \arrow[r, "*", "\Delta\cup\Delta_{k'}"'] & C[f(q_1,\ldots,q_n)] \arrow[r, "*", "\rho"'] & C[q'] \arrow[r, "*", "\Delta_{k'+1}"'] & {[t]} \\
D[\ell\tau] \arrow[r, dotted, "*", "\mathcal{R}"'] & D[r\tau] \arrow[r, dotted, "*", "\Delta\cup\Delta_{k'}"'] & C[r\tau] \arrow[r, dotted, "*", "\Delta\cup\Delta_{k'}"'] & C[r\theta] \arrow[u, dotted, "*", "\Delta_{k'}"'] &
\end{tikzcd}$$

Fig. 4: Bypassing $\rho$ to close the induction step.

on the size of $\Delta_{k'+1} \setminus \Delta_{k'}$. If it is empty we have a $\Delta_{k'}$-sequence and may simply close the proof with an application of the first induction hypothesis. Otherwise we have some set $\Delta$ and transition $\rho\colon f(q_1,\ldots,q_n) \to q'$ that was created from some rule $\ell \to r \in \mathcal{R}$ with $\ell = f(\ell_1,\ldots,\ell_n)$ and the sequence $r\theta \to^*_{\Delta'_k} q'$ by an application of rule (†) such that $\{\rho\} \uplus \Delta \subseteq \Delta_{k'+1} \setminus \Delta_{k'}$. The second induction hypothesis is if $s \to^+_{\Delta\cup\Delta_{k'}} [t]$ then $s \in (\to^*_{\mathcal{R}})[\Sigma(t)]$. Let $m$ denote the number of steps that use $\rho$. We continue by a third induction on $m$. If $m = 0$ the sequence from $s$ to $[t]$ only used transitions in $\Delta \cup \Delta_{k'}$ and using the second induction hypothesis we are done. Otherwise there is some $m' \geqslant 0$ such that $m = m' + 1$ and the induction hypothesis is that for all terms $s, t$ if $s \to^+_{\Delta\cup\Delta_{k'}} [t]$ using $\rho$ only $m'$ times then $s \in (\to^*_{\mathcal{R}})[\Sigma(t)]$. Now we look at the first step using $\rho$ in the sequence, that is, $s = D[f(s_1,\ldots,s_n)] \to^*_{\Delta\cup\Delta_{k'}} C[f(q_1,\ldots,q_n)] \to_\rho C[q'] \to^*_{\Delta_{k'+1}} [t]$. Note that from this we get $D[u] \to^*_{\Delta\cup\Delta_{k'}} C[u]$ for all terms $u$.

Next we define a substitution $\tau$ such that

$$s \to^*_{\mathcal{R}} D[\ell\tau] \to_{\mathcal{R}} D[r\tau] \to^*_{\Delta\cup\Delta_{k'}} C[r\tau] \to^*_{\Delta\cup\Delta_{k'}} C[q']$$

This allows us to bypass the $\rho$-step and so we arrive at a $\Delta_{k'+1}$-sequence from $D[r\tau]$ to $[t]$ containing one less $\rho$-step as shown in Figure 4. The construction of $\tau$ proceeds as follows: We fix $1 \leqslant i \leqslant n$. If $\ell_i$ is a variable in $r$ define $\tau_i$ to be $\{\ell_i \mapsto s_i\}$. Otherwise we know from the definition of inference rule (†) that $q_i = [\ell_i]$ and $s_i \to^+_{\Delta\cup\Delta_{k'}} [\ell_i]$. From that we have that $s_i \in (\to^*_{\mathcal{R}})[\Sigma(\ell_i)]$ but that means that there is some substitution $\tau_i$ such that $s_i \to^*_{\mathcal{R}} \ell_i\tau_i$. Moreover let $\tau' = \{x \mapsto u_x \mid x \in \mathcal{V}(r) \setminus \mathcal{V}(\ell)\}$ where $u_x$ is an arbitrary but fixed ground term such that $u_x \to^*_{\Delta_0} x\theta$. Since all states in $\mathcal{A}_0$ are accessible we can always find such a term $u_x$. Now let $\tau$ be the disjoint union of $\tau_1,\ldots,\tau_n,\tau'$. This substitution is well-defined because $\ell$ is linear. By construction of $\tau$ we get $s \to^*_{\mathcal{R}} D[\ell\tau]$.

Consider a variable $x$ occurring in $r$. If $x$ also occurs in $\ell$ we have $x = \ell_i$ for some unique $1 \leqslant i \leqslant n$ because $\mathcal{R}$ is growing. But then by construction of $\tau_i$ we get $x\tau = \ell_i\tau_i = s_i$. Moreover from the definition of $q_i$ in inference rule (†) we have $q_i = \ell_i\theta = x\theta$. But then $x\tau \to^+_{\Delta\cup\Delta_{k'}} x\theta$ from $s_i \to^+_{\Delta\cup\Delta_{k'}} q_i$. On the other hand, if $x$ does not occur in $\ell$ then $x\tau = x\tau'$ and $x\tau' \to^*_{\Delta_0} x\theta$ by construction of $\tau'$. So in both cases $r\tau \to^*_{\Delta\cup\Delta_{k'}} r\theta$. Together with $r\theta \to^*_{\Delta_{k'}} q'$ and $C[q'] \to^*_{\Delta_{k'+1}} [t]$ we may construct the sequence $D[r\tau] \to^+_{\Delta_{k'+1}} q_f$ which uses $\rho$ only $m'$ times. Using the induction hypothesis we arrive at $D[r\tau] \in (\to^*_{\mathcal{R}})[\Sigma(t)]$. Together with $s \to^*_{\mathcal{R}} D[\ell\tau] \to^*_{\mathcal{R}} D[r\tau]$ this means that $s \in (\to^*_{\mathcal{R}})[\Sigma(t)]$ and we are done. ◻

**Lemma 24 (Non-reachability via anc)** *Let $\mathcal{R}$ be a linear and growing TRS over signature $\mathcal{F}$. We may conclude non-reachability of $t$ from $s$ if*

$$L(\mathcal{A}_{\Sigma(s)}) \cap L(\mathsf{anc}_{\mathcal{R}}(\mathcal{A}_{\Sigma(t)})) = \varnothing$$ ☑

To see this in action look at the following example featuring Cops #494 [30, Example 16]:

*Example 23 (Infeasibility via* anc*, Cops* #494*)* Consider the CTRS $\mathcal{R}$ consisting of the following five rules:

$$\mathsf{g}(\mathsf{a}, x) \to \mathsf{c} \Leftarrow \mathsf{f}(x, \mathsf{a}) \twoheadrightarrow \mathsf{a} \qquad \mathsf{f}(\mathsf{a}, x) \to \mathsf{a} \qquad \mathsf{c} \to \mathsf{c}$$
$$\mathsf{g}(x, \mathsf{a}) \to \mathsf{d} \Leftarrow \mathsf{f}(x, \mathsf{b}) \twoheadrightarrow \mathsf{b} \qquad \mathsf{f}(\mathsf{b}, x) \to \mathsf{b}$$

It has two critical pairs

$$\mathsf{c} \approx \mathsf{d} \Leftarrow \mathsf{f}(\mathsf{a}, \mathsf{b}) \twoheadrightarrow \mathsf{b}, \mathsf{f}(\mathsf{a}, \mathsf{a}) = \mathsf{a}$$

and the symmetric one. Since $\mathsf{tcap}(\mathsf{f}(\mathsf{a}, \mathsf{b})) = x \sim \mathsf{b}$ and $\mathsf{tcap}(\mathsf{f}(\mathsf{a}, \mathsf{a})) = x \sim \mathsf{a}$ as well as $\mathsf{f} \sqsupset^{+}_{\mathsf{stg}} \mathsf{b}$ and $\mathsf{f} \sqsupset^{+}_{\mathsf{stg}} \mathsf{a}$ neither unification nor the symbol transition graph are sufficient to show infeasibility of these critical pairs. On the other hand, since the underlying TRS $\mathcal{R}_u$ is linear and growing, we may construct the tree automaton $\mathcal{A}_{\Sigma(\mathsf{f}(\mathsf{a},\mathsf{b}))}$ consisting of the three transitions

$$\mathsf{a} \to [\mathsf{a}] \qquad \mathsf{b} \to [\mathsf{b}] \qquad \mathsf{f}([\mathsf{a}], [\mathsf{b}]) \to [\mathsf{f}(\mathsf{a}, \mathsf{b})]$$

where $[\mathsf{f}(\mathsf{a}, \mathsf{b})]$ is the final state, as well as the tree automaton $\mathsf{anc}_{\mathcal{R}_u}(\mathcal{A}_{\Sigma(\mathsf{b})})$ consisting of 20 transitions

| | | | | |
|---|---|---|---|---|
| $\mathsf{a} \to \square$ | $\mathsf{a} \to [\mathsf{a}]$ | $\mathsf{g}(\square, \square) \to \square$ | $\mathsf{f}([\mathsf{b}], \square) \to \square$ | $\mathsf{g}(\square, [\mathsf{a}]) \to [\mathsf{g}(\square, \mathsf{a})]$ |
| $\mathsf{b} \to \square$ | $\mathsf{b} \to [\mathsf{b}]$ | $\mathsf{g}(\square, [\mathsf{a}]) \to \square$ | $\mathsf{f}([\mathsf{a}], \square) \to [\mathsf{a}]$ | $\mathsf{f}([\mathsf{a}], \square) \to [\mathsf{f}(\mathsf{a}, \square)]$ |
| $\mathsf{c} \to \square$ | $\mathsf{c} \to [\mathsf{c}]$ | $\mathsf{g}([\mathsf{a}], \square) \to \square$ | $\mathsf{f}([\mathsf{b}], \square) \to [\mathsf{b}]$ | $\mathsf{g}([\mathsf{a}], \square) \to [\mathsf{g}(\mathsf{a}, \square)]$ |
| $\mathsf{d} \to \square$ | $\mathsf{f}(\square, \square) \to \square$ | $\mathsf{f}([\mathsf{a}], \square) \to \square$ | $\mathsf{g}([\mathsf{a}], \square) \to [\mathsf{c}]$ | $\mathsf{f}([\mathsf{b}], \square) \to [\mathsf{f}(\mathsf{b}, \square)]$ |

with final state $[\mathsf{b}]$. Because the language of the intersection automaton is empty we have shown infeasibility of the condition $\mathsf{f}(\mathsf{a}, \mathsf{b}) \twoheadrightarrow \mathsf{b}$ by Lemma 24. So both critical pairs are infeasible.

In the setting of Section 4 the right-hand sides of conditions are always linear terms (because of right-stability; see Definition 1). Hence it is beneficial to start with the ground-instance automaton for the right-hand side of a condition (which in this case is exact) and compute the set of ancestors rather than taking the possibly non-linear left-hand side of a condition, overapproximating the ground-instances and only then computing the descendants of this set. Although this is not necessarily true in the setting of Section 5 (there we have no linearity-restriction on the right-hand sides of conditions) we employ the same setup for the sake of convenience.

For one of our main use cases, Theorem 5, we are restricted to left-linear CTRSs (via almost orthogonality) and linear right-hand sides of conditions (via right-stability). The latter also holds for right-hand sides that are combined by a compound symbol (again by right-stability). We show that in this setting anc subsumes tcap (at least in theory and for the forward direction).

**Lemma 25** *Let $\mathcal{R}$ be a left-linear CTRS and $t$ a linear term. If* tcap *can show non-reachability of $t$ from $s$, then so can* anc*.*

*Proof* Below we write $\mathsf{ren}(t)$ for a linearization of the term $t$ using fresh variables. We proof the contrapositive and assume that $\mathsf{anc}$ cannot show non-reachability. Moreover, let $\mathcal{R}'$ denote the result of applying the linear growing approximation to $\mathcal{R}_u$. Then there is some term $u$ such that $u \in L(\mathcal{A}_{\Sigma(s)})$ and $u \in L(\mathsf{anc}_{\mathcal{R}'}(\mathcal{A}_{\Sigma(t)}))$. Since $t$ is linear and $\mathcal{R}'$ is linear and growing the latter implies that $u \in (\rightarrow^*_{\mathcal{R}'})[\Sigma(t)]$ by Theorem 8 and thus $u \rightarrow^*_{\mathcal{R}'} t\tau$ for some substitution $\tau$. By Lemma 15, this means that $t\tau \in [\![\mathsf{tcap}_{\mathcal{R}'}(u)]\!]$. Since $u \in L(\mathcal{A}_{\Sigma(s)})$, it is clearly the case that $u \in \Sigma(\mathsf{ren}(s))$ and thus $u = \mathsf{ren}(s)\sigma$ for some substitution $\sigma$. Moreover $[\![\mathsf{tcap}_{\mathcal{R}'}(u)]\!] \subseteq [\![\mathsf{tcap}_{\mathcal{R}'}(\mathsf{ren}(s))]\!] = [\![\mathsf{tcap}_{\mathcal{R}'}(s)]\!]$. Together with $t\tau \in [\![\mathsf{tcap}_{\mathcal{R}'}(u)]\!]$ from above, we obtain $t\tau \in [\![\mathsf{tcap}_{\mathcal{R}'}(s)]\!]$. However, $\mathsf{tcap}$ does only consider the left-hand sides of rules, which are the same in $\mathcal{R}'$ and $\mathcal{R}_u$, thus also $t\tau \in [\![\mathsf{tcap}_{\mathcal{R}_u}(s)]\!]$ which implies $\mathsf{tcap}_{\mathcal{R}_u}(s) \sim t$. $\square$

If we also consider the reverse direction, that is, checking if the term $s$ is reachable from $t$ by $\mathcal{R}_u^{-1}$ for some condition $s \twoheadrightarrow t$ in Theorem 5, then $\mathsf{tcap}$ may well succeed where $\mathsf{anc}$ fails, as shown by the next example featuring Cops #546 [30, Example 23].

*Example 24 (anc vs. tcap, Cops #546)* The oriented 3-CTRS $\mathcal{R}$ consisting of the two rules

$$\mathsf{g}(x) \rightarrow \mathsf{g}(x) \Leftarrow \mathsf{g}(x) \twoheadrightarrow \mathsf{f}(\mathsf{a},\mathsf{b}) \qquad\qquad \mathsf{g}(x) \rightarrow \mathsf{f}(x,x)$$

is right-stable and extended properly oriented. It has two symmetric CCPs of the form

$$\mathsf{f}(x,x) \approx \mathsf{g}(x) \Leftarrow \mathsf{g}(x) \twoheadrightarrow \mathsf{f}(\mathsf{a},\mathsf{b}).$$

The underlying TRS $\mathcal{R}_u$ is not linear and growing, so if we want to apply $\mathsf{anc}$ we have to apply a linear growing approximation, resulting for example in $\mathcal{R}'$

$$\mathsf{g}(x) \rightarrow \mathsf{f}(x,y) \qquad\qquad \mathsf{g}(x) \rightarrow \mathsf{g}(x)$$

But then $\mathsf{anc}$ is not able to show infeasibility since the language accepted by the tree automaton $\mathcal{A}_{\Sigma(\mathsf{g}(x))} \cap \mathsf{anc}_{\mathcal{R}'}(\mathcal{A}_{\Sigma(\mathsf{f}(\mathsf{a},\mathsf{b}))})$ is not empty and also for the reverse direction $\mathcal{A}_{\Sigma(\mathsf{f}(\mathsf{a},\mathsf{b}))} \cap \mathsf{anc}_{\mathcal{R}'^{-1}}(\mathcal{A}_{\Sigma(\mathsf{g}(x))})$ we get a non-empty language. On the other hand, using the reversed underlying system $\mathcal{R}_u^{-1}$

$$\mathsf{f}(x,x) \rightarrow \mathsf{g}(x) \qquad\qquad \mathsf{g}(x) \rightarrow \mathsf{g}(x)$$

we have that $\mathsf{tcap}_{\mathcal{R}_u^{-1}}(\mathsf{f}(\mathsf{a},\mathsf{b})) = \mathsf{f}(\mathsf{a},\mathsf{b}) \not\sim \mathsf{g}(x)$. So in this case $\mathsf{tcap}$ succeeds where $\mathsf{anc}$ fails.

## 7.5 Exploiting Equalities

Finally, there are four cases where the equality between some terms in the conditions can be exploited to check a CCP for infeasibility. These four situations are depicted in Figure 5 and explained below.

Assume we have a CCP $\ell \approx r \Leftarrow c$ where $c$ contains at least two different conditions $s \twoheadrightarrow t$ and $u \twoheadrightarrow v$. Now $s = v$ implies that for $c$ to be satisfiable $t$ has to be reachable from $u$ (see Figure 5(a)). So if we can show that $t$ is not reachable

$$\begin{tikzcd} & s = v \arrow[dl, "*"'] & \\ t & & u \arrow[ul, "*"'] \end{tikzcd}$$

(a) $u \to^* t$

$$\begin{tikzcd} & t = u \arrow[dr, "*"] & \\ s \arrow[ur, "*"] & & v \end{tikzcd}$$

(b) $s \to^* v$

$$\begin{tikzcd} & t = v & \\ s \arrow[ur, "*"] & & u \arrow[ul, "*"'] \end{tikzcd}$$

(c) $s \downarrow u$

$$\begin{tikzcd} & s = u \arrow[dl, "*"'] \arrow[dr, "*"] & \\ t & & v \end{tikzcd}$$

(d) $t \downarrow v$

Fig. 5: Requirements for the conditions $c$ containing $s \twoheadrightarrow t$ and $u \twoheadrightarrow v$ to be satisfiable.

from $u$ we know that the conditions $c$ are infeasible. Similarly for $t = u$ to have any chance to satisfy the conditions $v$ has to be reachable from $s$ otherwise $c$ is infeasible (see Figure 5(b)). On the other hand if $t = v$ then to satisfy the conditions $s$ and $u$ have to be joinable and so from non-joinability of $s$ and $u$ we can conclude infeasibility of $c$ (see Figure 5(c)). Finally, if we are in the setting described in Section 4 then if $s = u$ to satisfy the conditions there would have to exist a diverging situation (see Figure 5(d)) but by the assumption of Definition 4.3 this implies that there is a join between $t$ and $v$ so to proof infeasibility of $c$ it suffices to show non-joinability of $t$ and $v$.

## 7.6 Certification

In this section we give an overview of all infeasibility and non-reachability techniques that are supported by our certifier CeTA and what kind of information it requires from a certificate in CPF [33] (short for *certification problem format*). Before we come to the special infeasibility condition of Definition 4, we handle the common case where, given a list of conditions $c$, we are interested in proving $\sigma, n \nvdash c$ for every substitution $\sigma$ and level $n$.

**Lemma 26 (Infeasibility certificates)** *Given $(\mathcal{R}, c)$ consisting of a CTRS $\mathcal{R}$ and a list of conditions $c = s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k$, infeasibility of $c$ with respect to $\mathcal{R}$ can be certified in one of the following ways:* ☑

1. *Provide a fresh function symbol $\mathsf{cs}$ of arity $n$ together with a non-reachability certificate for $(\mathcal{R}_u, \mathsf{cs}(s_1, \ldots, s_k), \mathsf{cs}(t_1, \ldots, t_k))$.*
2. *Provide two terms $s$ and $t$ with $s \twoheadrightarrow t \in c$, and a non-reachability certificate for $(\mathcal{R}_u, s, t)$.*
3. *For an arbitrary subset $c'$ of $c$, provide an infeasibility certificate for $(\mathcal{R}, c')$.*
4. *Provide three terms $s$, $t$, and $u$ such that $s \twoheadrightarrow u$ and $t \twoheadrightarrow u$ are equations in $c$ together with a non-joinability certificate for $(\mathcal{R}_u, s, t)$.*
5. *Provide three terms $s$, $t$, and $u$ such that $s \twoheadrightarrow t$ and $t \twoheadrightarrow u$ are equations in $c$ together with a non-reachability certificate for $(\mathcal{R}_u, s, u)$.*

*Proof* Note that 3 is obvious and 1 only exists for tool-author convenience but is subsumed by the combination of 2 and 3. Moreover, 2 follows from the fact that $\mathsf{cs}(s_1, \ldots, s_k)\sigma \not\to^*_{\mathcal{R}_u} \mathsf{cs}(t_1, \ldots, t_k)\tau$ for all $\sigma$ and $\tau$, implies the existence of at least one $1 \leqslant i \leqslant k$ such that $s_i\sigma \not\to^*_{\mathcal{R}_u} t_i\tau$ for all $\sigma$ and $\tau$. Finally, for 4, whenever $s\sigma$ and $t\tau$ are not joinable for arbitrary $\sigma$ and $\tau$, the existence of $\mu$ and $n$ such that $\mu, n \vdash s \twoheadrightarrow u,\ t \twoheadrightarrow u$ is impossible. □

Note that in 2 we check for non-reachability between left-hand sides and their corresponding right-hand sides, while in 4 we check for non-joinability between two left-hand sides. Thus, while in general non-joinability is more difficult to show than non-reachability, 4 is not directly subsumed by 2.

**Lemma 27 (Non-reachability certificates)** *Given $(\mathcal{R}, s, t)$ consisting of a TRS $\mathcal{R}$ and two terms $s$ and $t$, $\mathcal{R}$-non-reachability of $t$ from $s$ can be certified in one of the following ways:* ☑

1. *Indicate that $\mathsf{tcap}(s)$ does not unify with $t$.*
2. *Indicate that generalized $\mathsf{tcap}$ shows non-reachability of $s$ from $t$.*
3. *Provide a TRS $\mathcal{R}'$ such that for each $\ell \to r \in \mathcal{R}$ there is $\ell' \to r' \in \mathcal{R}'$ and a substitution $\sigma$ with $\ell = \ell'\sigma$ and $r = r'\sigma$, together with a non-reachability certificate for $(\mathcal{R}', s, t)$.*
4. *Provide a non-reachability certificate for $(\mathcal{R}^{-1}, t, s)$.*
5. *Use ordered completion to conclude non-reachability [34].*
6. *Make sure that $\mathcal{R}$ is linear and growing and provide a finite signature $\mathcal{F}$ and two constants $\mathsf{a}$ and $\square$ such that $\mathsf{a} \in \mathcal{F}$ but $\square \notin \mathcal{F}$, together with a tree automaton $\mathcal{A}$ that is an overapproximation of $\mathsf{anc}_{\mathcal{R}}(\mathcal{A}_{\Sigma(t)})$ and that satisfies $L(\mathcal{A}_{\Sigma(s)}) \cap L(\mathcal{A}) = \varnothing$.*

*Proof* If $\mathsf{tcap}_{\mathcal{R}}(s) \not\sim t$, then 1 holds by Lemma 15. The same holds for 2 using Corollary 4. Further note that $\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}'}$ and thus 3 is immediate. Moreover, 4 is obvious and 5 is shown by Sternagel and Winkler [34], leaving us with 6. From a certification perspective this is the most interesting case. To begin with, there are two reasons why we do not want to repeat the full construction of $\mathsf{anc}$ inside CeTA. Firstly, we would unnecessarily repeat an operation with at least exponential worst-case complexity. Secondly, a fully-verified executable algorithm is not even part of our formalization, instead we heavily rely on inductive definitions. While turning the existing inductive definitions into executable recursive functions would definitely be possible, we stress that this is not necessary. In CeTA we check that $\mathcal{A}$ is an overapproximation of $\mathsf{anc}_{\mathcal{R}}(\mathcal{A}_{\Sigma(t)})$ as follows: firstly, we ensure that $\mathcal{A}$ does not contain epsilon transitions, that $[t]$ is included in the final states of $\mathcal{A}$, and that $\Delta_t$ as well as the matching rules with respect to the signature $\mathcal{F}$ are part of the transitions of $\mathcal{A}$; secondly, we check that $\mathcal{A}$ is closed with respect to inference rule (†). Since $\mathcal{A}_{\Sigma(s)}$ is an overapproximation of $\Sigma(s)$ and by the required conditions together with Theorem 8, $L(\mathcal{A})$ overapproximates $[\to^*_{\mathcal{R}}](\Sigma(t))$, we can conclude $\Sigma(s) \cap [\to^*_{\mathcal{R}}](\Sigma(t)) = \varnothing$. Thus there are no *ground* substitutions $\sigma$ and $\tau$ such that $s\sigma, t\tau \in \mathcal{T}(\mathcal{F})$ and $s\sigma \to^*_{\mathcal{R}} t\tau$. In order to conclude that the same holds true for *arbitrary* substitutions (not necessarily restricted to $\mathcal{F}$), we rely on an earlier result [32] that implies that whenever $s\sigma \to^*_{\mathcal{R}} t\tau$ for arbitrary $\sigma$ and $\tau$ and $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ then there are $\sigma'$ and $\tau'$ such that $s\sigma', t\tau' \in \mathcal{T}(\mathcal{F})$ and $s\sigma' \to^*_{\mathcal{R}} t\tau'$. □

Note that 3 allows us to certify the linear growing approximation of a TRS without actually having to formalize it in Isabelle/HOL. More specifically, whenever $\mathcal{R}'$ is the result of applying the linear growing approximation to $\mathcal{R}$, then the corresponding certificate will pass 3 and $\mathcal{R}'$ will be linear and growing in the check for 6; otherwise 6 will fail.

**Lemma 28 (Non-joinability certificates)** *Given $(\mathcal{R}, s, t)$ consisting of a TRS $\mathcal{R}$ and two term $s$ and $t$, $\mathcal{R}$-non-joinability of $s$ and $t$ can be certified in one of the following ways:* ☑

1. *Indicate that* $\mathsf{tcap}(s)$ *does not unify with* $\mathsf{tcap}(t)$*.*
2. *If at least one of the terms, say* $t$*, is a ground* $\mathcal{R}$*-normal form provide a non-reachability certificate for* $(\mathcal{R}, s, t)$*.*

*Proof* We prove 1 by Lemma 16 and 2 by Lemma 15 since non-joinability reduces to non-reachability when one of the terms is an $\mathcal{R}$-normal form. □

**Lemma 29 (Ao-infeasibility certificates)** *Given the triple* $(\mathcal{R}, c_1, c_2)$ *consisting of a CTRS* $\mathcal{R}$ *fulfilling all syntactic requirements of Theorem 5 and two lists of conditions* $c_1$ *and* $c_2$*, infeasibility with respect to almost orthogonality can be certified in one of the following ways:* ☑

1. *Provide an infeasibility certificate for* $(\mathcal{R}, c)$ *where* $c$ *is the concatenation of* $c_1$ *and* $c_2$*.*
2. *Provide three terms* $s$*,* $t$ *and* $u$ *such that* $s \twoheadrightarrow t$ *is an equation in* $c_1$ *and* $s \twoheadrightarrow u$ *an equation in* $c_2$*, together with a non-joinability certificate for* $(\mathcal{R}_u, t, u)$*.*

*Proof* While 1 follows from Lemma 26, in 2 we make use of the level-commutation assumption of Definition 4 to deduce non-meetability of the terms $t$ and $u$ from non-joinability of $t$ and $u$. □

The formalization of the methods described in this section can be found in the following IsaFoR theory files:

```
thys/Rewriting/          thys/Tree_Automata/
   Tcap.thy                 Exact_Tree_Automata_Completion.thy
   Ground_Context.thy       Exact_Tree_Automata_Completion_Impl.thy

thys/Nonreachability/    thys/Conditional_Rewriting/
   Gtcap.thy                Infeasibility.thy
   Gtcap_Impl.thy           Level_Confluence_Impl.thy
   Nonreachability.thy
```

In the next section we will look at two supporting methods that facilitate the techniques described in this and the previous sections.

## 8 Supporting Methods

Sometimes directly using the methods for (non-)confluence that are described in earlier sections is not possible. But there are sound methods that can "simplify" a given CTRS and make it amenable for them. In this section we will see two such methods.

The first one is about infeasibility. Already in Section 7 we have seen that infeasibility of CCPs can be expedient in analyzing confluence of CTRSs. Here we show that also infeasibility of conditions of rewrite rules can be beneficial for confluence analysis.

The second method can "reshape" certain conditional rewrite rules in such a way that other methods become more applicable.

## 8.1 Infeasible Rule Removal

If the conditions of a conditional rewrite rule are infeasible we can just remove this rule from the CTRS without changing the rewrite relation because the rule could never be fired anyway.

**Definition 21 (Infeasible rule)** A conditional rewrite rule $\ell \to r \Leftarrow c$ is called *infeasible* if $c$ is infeasible.

You might ask, why anyone would add an infeasible rule to a CTRS in the first place? On the one hand, it might be a bug by a programmer. On the other hand, there are transformations from programs to CTRSs that occasionally result in infeasible rules. In fact, when looking into the Cops database, we find that from 111 DCTRSs a stunning 11 systems contain rules that we can show infeasible. And that is just using fast and easy checks so there might be more where infeasibility of the conditions is not so obvious. All 11 DCTRSs with infeasible rules are from the literature. For instance the example below is due to Bergstra and Klop.

*Example 25* Remember the DCTRS $\mathcal{R}$ from Example 13 consisting of the following four rules

$$\mathsf{p}(\mathsf{q}(x)) \to \mathsf{p}(\mathsf{r}(x)) \qquad \mathsf{q}(\mathsf{h}(x)) \to \mathsf{r}(x) \qquad \mathsf{r}(x) \to \mathsf{r}(\mathsf{h}(x)) \Leftarrow \mathsf{s}(x) \twoheadrightarrow 0 \qquad \mathsf{s}(x) \to 1$$

The right-hand side 0 of the single condition of the only conditional rule of $\mathcal{R}$ is an $\mathcal{R}$-normal form. The left-hand side $\mathsf{s}(x)$ does only rewrite to the different $\mathcal{R}$-normal form 1. Hence the condition is infeasible and we can just remove the rule, because it does not affect the rewrite relation of $\mathcal{R}$ in any way.

In principle we can use all the methods from Section 7 to remove infeasible rules before we employ any of the actual (non-)confluence checks. In the following example we utilize $\mathsf{anc}$ (see Section 7.4).

*Example 26* Consider the CTRS $\mathcal{R}$ consisting of the two rules

$$\mathsf{h}(x) \to \mathsf{a} \qquad\qquad \mathsf{g}(x) \to \mathsf{a} \Leftarrow \mathsf{h}(x) \twoheadrightarrow \mathsf{b}$$

The condition of the only conditional rewrite rule is infeasible because $\mathsf{h}(x)$ only rewrites to $\mathsf{a}$ and not to $\mathsf{b}$. Unification fails to show that because $\mathsf{tcap}(\mathsf{h}(x)) = y \sim \mathsf{b} = \mathsf{tcap}(\mathsf{b})$. Fortunately we have $\mathsf{h} \not\sqsupseteq_{\mathsf{rs}} \mathsf{b}$ and hence the condition $\mathsf{h}(x) \twoheadrightarrow \mathsf{b}$ is infeasible by Lemma 19. Or we can use exact tree automata completion. The underlying TRS $\mathcal{R}_u$ is linear and growing and hence we can construct the tree automata $\mathcal{A}_{\Sigma(\mathsf{h}(x))}$ and $\mathsf{anc}_{\mathcal{R}_u}(\mathcal{A}_{\Sigma(\mathsf{b})})$. The language of the intersection automaton is empty and the condition $\mathsf{h}(x) \twoheadrightarrow \mathsf{b}$ is infeasible by Lemma 24.

Another useful result, due to Sternagel and Yamada [35, Theorem 3], that we formalized, is that we may ignore those rules of a CTRS that we are trying to show infeasible during our investigation:[11]

**Theorem 9** *A set of rules $\mathcal{S}$ is infeasible with respect to a CTRS $\mathcal{R}$ iff it is infeasible with respect to $\mathcal{R} \setminus \mathcal{S}$.* ☑

[11] This result is applied implicitly by CeTA whenever checking a certificate that claims to remove infeasible rules.

## 8.2 Inlining of Conditions

In this section we look at another simple method that is inspired by inlining of let-constructs and where-expressions in compilers. We give a transformation on CTRSs which is often helpful in practice.

**Definition 22 (Inlining of Conditions ✓)** Given a conditional rewrite rule

$$\rho\colon \ell \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_k \twoheadrightarrow t_k$$

and an index $1 \leqslant i \leqslant k$ such that $t_i = x$ for some variable $x$, let $\mathsf{inl}_i(\rho)$ denote the rule resulting from *inlining* the $i$-th condition of $\rho$, that is,

$$\ell \to r\sigma \Leftarrow s_1\sigma \twoheadrightarrow t_1, \ldots, s_{i-1}\sigma \twoheadrightarrow t_{i-1}, s_{i+1}\sigma \twoheadrightarrow t_{i+1}, \ldots, s_k\sigma \twoheadrightarrow t_k$$

with $\sigma = \{x \mapsto s_i\}$.

**Lemma 30** *Let $\rho \in \mathcal{R}$ and $s \twoheadrightarrow x$ be the $i$-th condition of $\rho$. Whenever we have $x \notin \mathcal{V}(\ell, s, t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_k)$, then for $\mathcal{R}' = (\mathcal{R} \setminus \{\rho\}) \cup \{\mathsf{inl}_i(\rho)\}$ the relations $\to_{\mathcal{R}}^*$ and $\to_{\mathcal{R}'}^*$ coincide.* ✓

*Proof* We show $\to_{\mathcal{R},n} \subseteq \to_{\mathcal{R}',n}^*$ and $\to_{\mathcal{R}',n} \subseteq \to_{\mathcal{R},n}$ by induction on the level $n$. For $n = 0$ the result is immediate. Consider a step $s = C[\ell\sigma] \to_{\mathcal{R},n+1} C[r\sigma] = t$ employing rule $\rho$ (for the other rules of $\mathcal{R}$ the result is trivial). Thus, $u\sigma \to_{\mathcal{R},n}^* v\sigma$ for all $u \twoheadrightarrow v \in c$. In particular $s\sigma \to_{\mathcal{R},n}^* x\sigma$. Thus, using the IH, for each condition $u \twoheadrightarrow v$ of $\mathsf{inl}_i(\rho)$ we have $u\sigma = s_j\{x \mapsto s\}\sigma \to_{\mathcal{R}',n}^* s_j\sigma \to_{\mathcal{R}',n}^* t_j\sigma = v\sigma$ for some $1 \leqslant j \leqslant k$. Hence, $\ell\sigma \to_{\mathcal{R}',n+1} r\{x \mapsto s\}\sigma \to_{\mathcal{R}',n+1}^* r\sigma$ and thus $s \to_{\mathcal{R}',n+1}^* t$.

Now, consider a step $s = C[\ell\sigma] \to_{\mathcal{R}',n+1} C[r\{x \mapsto s\}\sigma]$ employing rule $\mathsf{inl}_i(\rho)$. Together with the IH this implies that $u\sigma \to_{\mathcal{R},n}^* v\sigma$ for all conditions $u \twoheadrightarrow v$ in $\mathsf{inl}_i(\rho)$. Let $\tau$ be a substitution such that $\tau(x) = s\sigma$ and $\tau(y) = \sigma(y)$ for all $y \neq x$. We have $s_i\tau = s\tau = x\tau = t_i\tau$ and $s_j\tau = s_j\{x \mapsto s\}\sigma \to_{\mathcal{R},n}^* t_j\sigma = t_j\tau$ for all $1 \leqslant j \leqslant k$ with $i \neq j$, since $x$ occurs neither in $s$ nor in the right-hand sides of conditions in $\mathsf{inl}_i(\rho)$. Therefore, $u \to_{\mathcal{R},n}^* v$ for all $u \twoheadrightarrow v \in c$. In summary, we have $s = C[\ell\sigma] = C[\ell\tau] \to_{\mathcal{R},n+1} C[r\tau] = C[r\{x \mapsto s\}\sigma]$, concluding the proof. □

We are not aware of any mention of this simple method in the literature, but found that in practice, exhaustive application of inlining increases the applicability of other methods like infeasibility via tcap and non-confluence via plain rewriting: for the former inlining yields more term structure, which may prevent tcap from replacing a subterm by a fresh variable and thus makes non-unifiability more likely; while for the latter inlining may yield CCPs without conditions and thereby make them amenable to non-joinability techniques for plain term rewriting [44] as witnessed by Cops #551.

*Example 27 (Cops #551)* Consider the quasi-decreasing ADCTRS $\mathcal{R}$ consisting of the following six rules:

$$\mathsf{min}(x : \mathsf{nil}) \to x \quad (1) \qquad x < 0 \to \mathsf{false} \quad (4)$$

$$\mathsf{min}(x : xs) \to x \Leftarrow \mathsf{min}(xs) \twoheadrightarrow y,\ x < y \twoheadrightarrow \mathsf{true} \quad (2) \qquad 0 < \mathsf{s}(y) \to \mathsf{true} \quad (5)$$

$$\mathsf{min}(x : xs) \to y \Leftarrow \mathsf{min}(xs) \twoheadrightarrow y,\ x < y \twoheadrightarrow \mathsf{false} \quad (3) \qquad \mathsf{s}(x) < \mathsf{s}(y) \to x < y \quad (6)$$

$\mathcal{R}$ has 6 CCPs, 3 modulo symmetry:

$$x \approx x \Leftarrow \mathsf{min}(\mathsf{nil}) \twoheadrightarrow y,\ x < y \twoheadrightarrow \mathsf{true} \tag{1,2}$$
$$x \approx y \Leftarrow \mathsf{min}(\mathsf{nil}) \twoheadrightarrow y,\ x < y \twoheadrightarrow \mathsf{false} \tag{1,3}$$
$$x \approx y \Leftarrow \mathsf{min}(xs) \twoheadrightarrow z,\ x < z \twoheadrightarrow \mathsf{true},\ \mathsf{min}(xs) \twoheadrightarrow y,\ x < y \twoheadrightarrow \mathsf{false} \tag{2,3}$$

To conclude confluence of the system it remains to check its CCPs. The first one, (1,2), is trivially context-joinable because the left- and right-hand sides coincide. Unfortunately, the methods used in ConCon are not able to handle either of the CCPs (1,3) and (2,3). So we are not able to conclude confluence of $\mathcal{R}$ just yet. But Rules (2) and (3) of $\mathcal{R}$ are both susceptible to inlining of conditions. For each of them, we may remove the first condition and replace $y$ by $\mathsf{min}(xs)$ resulting in

$$\mathsf{min}(x : xs) \to x \Leftarrow x < \mathsf{min}(xs) \approx \mathsf{true} \tag{2'}$$
$$\mathsf{min}(x : xs) \to \mathsf{min}(xs) \Leftarrow x < \mathsf{min}(xs) \approx \mathsf{false} \tag{3'}$$

Now we actually arrive at the CTRS from Example 11 which is shown to be confluent in Section 5.

## 8.3 Certification and Implementation

Infeasible rule removal and inlining of conditions are both implemented in ConCon as a preprocessing step and are certifiable by CeTA.

For the certification of infeasible rule removal we can reuse machinery in ConCon and CeTA that is also used to show infeasibility of CCPs. The certificate just has to provide the infeasible rules together with the infeasibility proofs for their conditions. Most of the time CTRSs do not contain infeasible rules (see Section 9) and some of the available infeasibility checks are rather expensive (especially tree automata techniques). So in our implementation we only employ the *symbol transition graph* (see Section 7.2) method in this case because it is fast and lightweight.

To certify inlining CeTA requires ConCon to output the inlined CTRS $\mathcal{R}'$ together with a list of pairs that in the first component contain all modified rules and in the second component the corresponding list of conditions that have been inlined in this rule. Internally CeTA employs this information to reverse the inlining and finally checks if the result corresponds to the original input CTRS $\mathcal{R}$.

The formalization of the methods described in this section can be found in the following IsaFoR theory files:

```
thys/Conditional_Rewriting/         thys/Conditional_Rewriting/
   Infeasibility.thy                   Inline_Conditions_Impl.thy
   Inline_Conditions.thy
```

In the next section we present the results of our extensive experiments comparing the different (non-)confluence and infeasibility methods on the confluence problems database.

## 9 Experiments

In the previous sections we have established the theory underlying several confluence and infeasibility methods and gave an overview of our corresponding Isabelle/HOL formalization as part of IsaFoR.

In the following, we compare these methods experimentally in order to get empirical evidence on their strengths and weaknesses. To this end we implemented the methods supported by CeTA in the CTRS confluence tool ConCon.[12] The tool is implemented in Scala, an object-functional programming language that runs on the Java virtual machine.

ConCon also implements two methods that are not certifiable by CeTA, specifically a variant of Theorem 1 (see [14]) relying on the notion of weak left-linearity of a DCTRS [13] and the inductive symbol transition graph due to Sternagel and Yamada [35]. For more details on ConCon see [36].

All experiments have been carried out on a 64bit GNU/Linux machine with 12 Intel® Core™ i7-5930K processors clocked at 3.50GHz and 32GB of RAM. The kernel version is `4.9.0-9-amd64`. The version of Java on this machine is 1.8.0_222. We had to increase the stack size used by ConCon to 20MB using the JVM flag '`-Xss20M`' to prevent stack overflows caused by parsing deep terms like in Cops #313.

### 9.1 Comparing ConCon's Confluence Methods

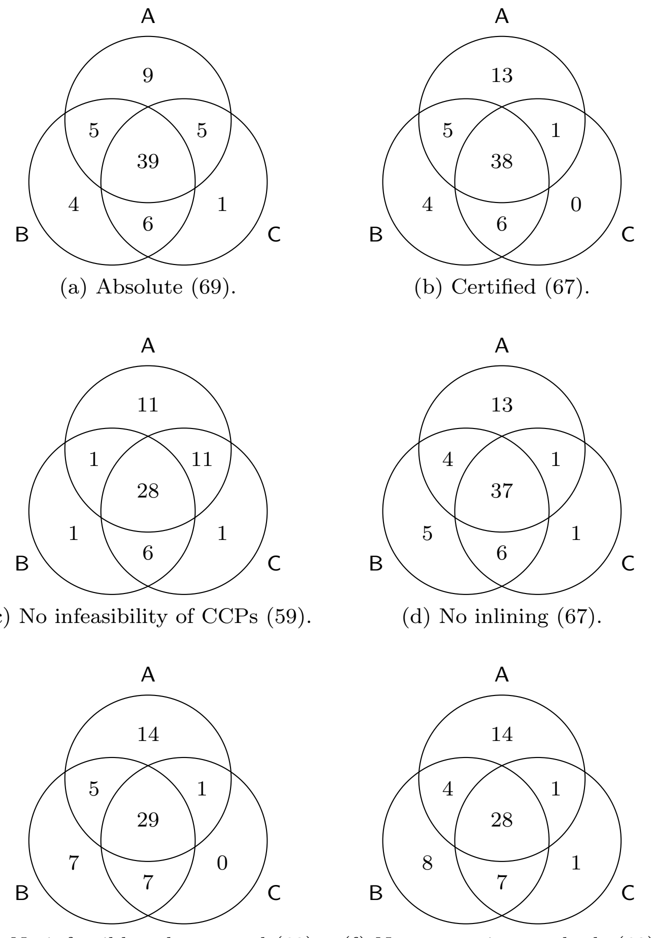
In this first part of our experiments we take a closer look at ConCon's confluence methods in comparison to each other. We use the 149 oriented CTRSs from Cops[13] version 1137, and set the timeout to one minute. The subfigures of Figure 6 compare the three confluence methods Theorem 6 (A), Theorem 2 (B), and Theorem 1 (C) for various settings of ConCon.

If we look at ConCon's absolute power, employing the supporting methods from Section 8, infeasibility of CCPs from Section 7, and also uncertified methods (which are not part of this article) we get the numbers in the Venn diagram depicted in Figure 6(a).

There is a total of 69 systems that are shown to be confluent. On its own method A is the strongest, succeeding on 58 CTRSs, closely followed by method B, which succeeds on 54 CTRSs, and finally method C, which can show confluence of 51 systems. Moreover, there are 9 CTRSs where only method A succeeds, 4 where only method B succeeds, and 1 where only method C succeeds.

The results for restricting to the methods that can be certified by CeTA can be found in Figure 6(b). In this case, the total is reduced by two systems (#286 and #793) to 67 confluent systems when compared to Figure 6(a). More specifically, methods A (57 total) and B (53 total) lose one system each, and method C six systems (45 total). The system lost by methods A and B is Cops #793. In both cases, it is lost because the certified infeasibility methods are not able to conclude that the first rule is infeasible. Concerning method C, Cops #286, #326, #319, #362, #793, and #806 are all weakly-left linear (and a variant of Theorem 1 for weakly-left linear CTRSs succeeds if we use uncertified methods) but their unravelings are not

[12] http://cl-informatik.uibk.ac.at/software/concon
[13] http://cops.uibk.ac.at?q=1..1137+oriented+ctrs



(a) Absolute (69). (b) Certified (67).

(c) No infeasibility of CCPs (59). (d) No inlining (67).

(e) No infeasible rule removal (63). (f) No supporting methods (63).

Fig. 6: Comparison of ConCon's confluence methods.

left-linear. Hence our formalization of Theorem 1 is not applicable. Of these six Cops #286 is the system where in Figure 6(a) only method C succeeds.

Next, in Figure 6(c), we see the confluence results when switching off all infeasibility checks for conditional critical pairs. The total number of systems that can be shown to be confluent is reduced by eight (#288, #292, #326, #340, #361, #494, #551, and #552) to 59 when compared to Figure 6(b). As expected—because it does not profit from infeasible CCPs—method C is not effected by this change. Method A loses six systems: #288, #292, #326, #340, #551, and #552. (While it does rely on infeasibility of CCPs, it also employs context-joinability and unfeasibility of CCPs (see Definition 7).) Method B is affected the most, losing 17 systems, because without infeasibility of CCPs method B is reduced to a purely syntactic check that relies on the absence of non-trivial critical pairs.

Switching off inlining (but still using infeasible rule removal) does not change the overall number of confluent systems (see Figure 6(d)) compared to Figure 6(b). However, when looking at the specific systems that could be shown to be confluent, we observe that we actually gained #529 (which times out without inlining) but lost #551. Recall that inlining can give terms more structure, which is good to show non-reachability using tcap. On the one hand, for Cops #551 without inlining ConCon's infeasibility methods are not able to handle all of the CCPs (as explained in Example 27). On the other hand, if we need tree automata completion to show infeasibility of a conditional critical pair more structure may lead to a larger tree automaton that we have to compute and hence to a timeout. Apparently this is what happens for Cops #529: while ConCon times out when inlining is used it succeeds when we switch it off. We also want to remark that although without inlining ConCon is able to produce a certificate for Cops #529, CeTA is actually not able to certify it because to do so it would have to compute $21^{11}$ state substitutions to check the given tree automaton and consequently we run out of memory. Besides, Cops #493 is now only proven confluent by method C because without inlining ConCon's infeasibility checks do not succeed on two of its CCPs.

Now, if we switch off infeasible rule removal (but keep inlining) the overall number of confluent systems is reduced by four to 63 when compared to Figure 6(b) (see Figure 6(e)). The lost systems are #317, #792, #794, and #806. Here methods A (49 total) and B (37 total) lose eight systems each, while method B (48 total) loses five systems.

For sake of completeness we also include Figure 6(f) which shows the results when switching off both inlining and infeasible rule removal. As expected the results are basically a combination of the previous two diagrams.

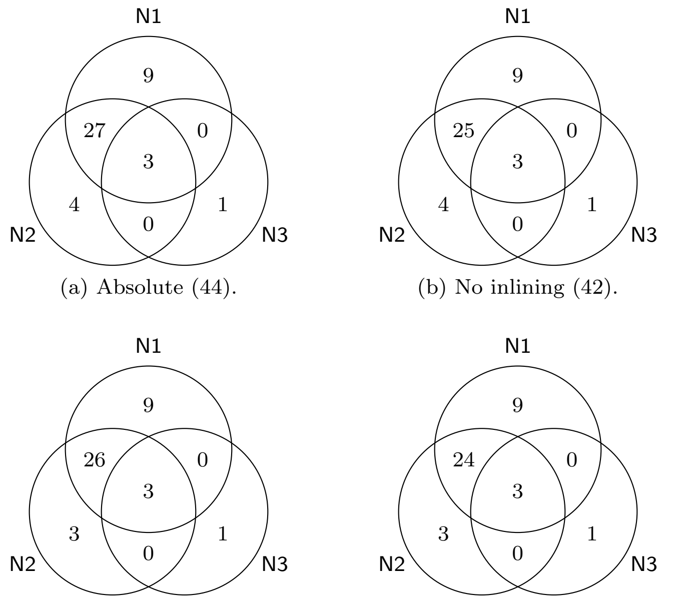
## 9.2 Comparing ConCon's Non-Confluence Methods

In this section we take a closer look at ConCon's non-confluence methods in comparison to each other. We use the same 149 oriented CTRSs as in the previous section and the same timeout of one minute. Results for the three non-confluence methods Lemma 14 (N1), Lemma 13 (N2), and Lemma 12 (N3) of ConCon are summarized in Figure 7.

When it comes to non-confluence ConCon is able to handle 44 systems when employing the supporting methods from Section 8. Figure 7(a) compares the three non-confluence methods of ConCon. On its own method N1 is the strongest succeeding on 39 systems, followed by method N2 succeeding on 34 systems, and finally method N3 which is specifically targeted at 4-CTRSs and hence can only handle the 4 systems of this kind.

As shown in Figure 7(b) when we switch off inlining method N2 loses 2 systems. Inlining transforms Cops #351 to an unconditional system and cannot be handled by ConCon without this method. Cops #353 can only be handled by method N1 without inlining. The total number of non-confluent systems is reduced to 41.

Similarly, as shown in Figure 7(c) when switching off infeasible rule removal method N2 loses one system. This system, Cops #271, is reduced to a TRS when employing infeasible rule removal and cannot be handled by ConCon without this method.



(a) Absolute (44). (b) No inlining (42).

(c) No infeasible rule removal (42). (d) No supporting methods (40).

Fig. 7: Comparison of ConCon's non-confluence methods.

Finally, the results for switching off both supporting methods are depicted in Figure 7(d). The total number of non-confluent systems is reduced to 40 because as explained earlier we lose Cops #351 and #271. Unlike in the case of the confluence methods, where inlining also causes ConCon to lose one system, in the non-confluence case it is exclusively beneficial.

## 9.3 Comparing ConCon's Infeasibility Methods

In this last section regarding our experiments we compare the five infeasibility methods of ConCon, that is, the tcap check (tcap; see Section 7.1, the generalized tcap check with symbol transition graph and decomposition of reachability problems (gtcap; see Section 7.2 and and Section 7.3), exploiting equalities in conditions (exeq; see Section 7.5), equational reasoning using MædMax [34] (oc; not part of this article), and finally, exact tree automata completion (etac; see Section 7.4).

To this end we take the 133 oriented infeasibility problems from Cops[14] and apply ConCon's infeasibility methods separately with a timeout of 60 seconds. (As an aside: infeasibility problems are a part of Cops since the International Confluence Competition in 2019.) The results of this experiments are listed in Table 1. The first row, labeled "infeasible," lists the number of problems which could be shown to be infeasible for each method (the numbers in parentheses

[14] `http://cops.uibk.ac.at?q=1..1137+infeasibility` but without #1137 and #1136 which are both semi-equational and not oriented.

| | tcap | gtcap | exeq | oc | etac | total |
|---|---|---|---|---|---|---|
| infeasible | 19 (114) | 22 (111) | 4 (129) | 20 (113) | 35 (98) | 40 (93) |
| timeout | 0 | 0 | 0 | 18 | 40 | 40 |

Table 1: Infeasibility results on 133 oriented infeasibility problems from Cops.

indicate how many problems are still open for that column). The row labeled "timeout" gives the number of problems for which a timeout occurred. On its own method etac is the strongest showing infeasibility of 35 problems. However, it also is very computation intensive and causes the most timeouts (40). There are seven problems where only method etac is able to show infeasibility. Next, method gtcap shows 22 problems infeasible and has no timeouts. Unsurprisingly, it subsumes method tcap. Method oc shows 20 problems infeasible and causes 18 timeouts. There are 4 problems where only method oc is able to show infeasibility. Method exeq shows four problems infeasible and does not cause any timeouts.

In summary, all methods together succeed on 40 problems and time out on 40.

While, at least on this benchmark, oc and etac together subsume all other methods, in practice it is still important to first employ fast methods like exeq and gtcap, and only then expensive methods like oc and etac in order to avoid timeouts. For removal of infeasible rules during confluence proofs (Section 8.1), fast methods are even more important, since only a small fraction of the overall time can be spared on checking infeasibility of rules.

This brings us to the final section of this article where we reflect on the findings of all previous sections and lay out some possible future work.

## 10 Conclusion

In the last few sections we presented an overview of the confluence and infeasibility results for CTRSs that we have formalized in IsaFoR. Through code generation these are now available in the certifier CeTA.

We first presented two of the three main confluence methods available in CeTA: the result that almost orthogonal (modulo infeasibility), right-stable, and extended properly oriented CTRSs are confluent in Section 4 and that quasi-decreasing, strongly deterministic CTRSs are confluent if all their conditional critical pairs are joinable in Section 5. Section 6 presented some checks, most notably a method employing conditional narrowing, to find witnesses for non-confluence of CTRSs. The topic of Section 7 were several methods to check for non-reachability between terms. In our context these are used to get infeasibility of conditional critical pairs as well as conditional rules in order to make the methods of Sections 4 and 5, that both rely on critical pair analysis, more applicable. In detail the non-reachability checks are by unification, the symbol transition graph employing decomposition of reachability problems, exact tree automata completion, and the exploitation of certain equalities in the conditions. After that we touched on two supporting methods in Section 8. One that employs some of the results of Section 7 to get rid of infeasible rules and another that inlines certain conditions of rules. Finally, Section 9 presented our experiments on the confluence problems database. We first compared ConCon's three confluence methods to each other. Following that, we presented experimental results on ConCon's non-confluence methods. Finally, we investigated ConCon's infeasibility methods in some detail.

## 10.1 Formalization and Implementation

A lot of work went into the formalization of the presented results. Here is a rough impression of the involved effort: our formalization comprises 92 definitions, 37 recursive functions, and 518 lemmas with proofs, on approximately 10,000 lines of Isabelle code (in addition to everything that we could reuse from the IsaFoR library).

To give some additional measure on how much work went into the formalization we take a look at the *de Bruijn factor* of some of its parts. The de Bruijn factor has been defined by Freek Wiedijk[15] to be the quotient of the size of a formalization of a mathematical text and the size of its informal original. Because our formalization depends on a lot of prior results from IsaFoR the scope of a specific result is not so easy to define and hence somewhat arbitrary. Nevertheless, when comparing the textual description in this article to our formalization in IsaFoR (without the additional setup for code generation and CeTA's parser) we get the following (approximate) numbers: the results of Section 7.4 have a de Bruijn factor of 9.8, Theorem 5 has a de Bruijn factor of 4.2, and finally the de Bruijn factor of the results in Section 5 is 4.5.

There are some points of notice: the textual description of all of the above mentioned proofs to some extend contain diagrams to convince the reader of certain steps. Such diagrams are notoriously hard to formalize. Further, the results in Section 7.4 have been the second author's first larger IsaFoR-development and he was still trying to get to grips with Isabelle/HOL. So he probably did not exploit Isabelle/HOL's automatic methods to their full potential, for that reason these proofs are quite verbose, which explains the much higher de Bruijn factor (in comparison to the later developments).

But also the work put in the automatic tool ConCon, although somewhat paling in comparison to the formalization, was significant. ConCon 1.9.1 consists of approximately 14,000 lines of Scala code and has been very successful in the confluence competition. At the time of writing it won the CPF-CTRS category for confluence of CTRSs with proof certificates four times in a row and is (after four years) still the only tool for CTRSs to provide certifiable output. Moreover, ConCon can decide confluence of roughly 75% of the oriented CTRSs in Cops and certify approximately 98% of these with the help of CeTA.

## 10.2 Future Work

In our opinion the presented work satisfactorily discharges the goal to produce a reliable and automatic tool to check confluence of conditional term rewrite systems. Nevertheless, there are some open issues:

[15] http://www.cs.ru.nl/~freek/factor

*Infeasibility.* Currently 36 out of the 149 oriented CTRSs in Cops cannot be solved by ConCon 1.9.1. One of those is Cops #327 for computing the greatest common divisor of two natural numbers:

$$\begin{array}{rlcc}
\mathsf{gcd}(x,x) \to x & & x < \mathsf{0} \to \mathsf{false} & \mathsf{0} - \mathsf{s}(y) \to \mathsf{0} \\
\mathsf{gcd}(\mathsf{s}(x),\mathsf{0}) \to \mathsf{s}(x) & & \mathsf{0} < \mathsf{s}(y) \to \mathsf{true} & x - \mathsf{0} \to x \\
\mathsf{gcd}(\mathsf{0},\mathsf{s}(y)) \to \mathsf{s}(y) & & \mathsf{s}(x) < \mathsf{s}(y) \to x < y & \mathsf{s}(x) - \mathsf{s}(y) \to x - y \\
\mathsf{gcd}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{gcd}(x - y,\mathsf{s}(y)) & \Leftarrow y < x \twoheadrightarrow \mathsf{true} & & \\
\mathsf{gcd}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{gcd}(\mathsf{s}(x),y - x) & \Leftarrow x < y \twoheadrightarrow \mathsf{true} & &
\end{array}$$

Because the system is not left-linear Theorem 5 is not applicable. Its unraveling is not left-linear, so Theorem 1 is also not applicable. But Cops #327 is a quasi-decreasing ADCTRS, making Theorem 7 applicable in principle. It only remains to show joinability (or infeasibility for that matter) of its CCPs. The CTRS has three CCPs (modulo symmetry) of which we show two (because the conditions of the third are similar):

$$\mathsf{gcd}(\mathsf{s}(x),y - x) \approx \mathsf{gcd}(x - y,\mathsf{s}(y)) \Leftarrow y < x \twoheadrightarrow \mathsf{true},\ x < y \twoheadrightarrow \mathsf{true}$$
$$\mathsf{gcd}(x - x,\mathsf{s}(x)) \approx \mathsf{s}(x) \Leftarrow x < x \twoheadrightarrow \mathsf{true}$$

Under (the reasonable assumption) that $<$ is supposed to encode "strictly less than," these CCPs are obviously infeasible ($y$ cannot be strictly smaller and strictly greater than $x$ at the same time for the first CCP and $x$ cannot be strictly smaller than itself for the second one). Unfortunately, this cannot be shown by the methods presented in Section 7. By using $\mathcal{R}_{\mathsf{u}}$ (for example when approximating non-reachability) we open the door for inconsistencies:

$$\mathsf{s}(\mathsf{0}) \xleftarrow{*} \mathsf{gcd}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{0})) \xleftarrow{*} \mathsf{gcd}(\mathsf{s}(\mathsf{s}(\mathsf{0})),\mathsf{s}(\mathsf{0})) \xrightarrow{*} \mathsf{gcd}(\mathsf{0},\mathsf{s}(\mathsf{s}(\mathsf{0}))) \xrightarrow{*} \mathsf{s}(\mathsf{s}(\mathsf{0}))$$

and thus $\mathsf{gcd}(\mathsf{s}(\mathsf{s}(\mathsf{0})),\mathsf{s}(\mathsf{0})) < \mathsf{gcd}(\mathsf{s}(\mathsf{s}(\mathsf{0})),\mathsf{s}(\mathsf{0})) \to^* \mathsf{s}(\mathsf{0}) < \mathsf{s}(\mathsf{s}(\mathsf{0})) \to^* \mathsf{true}$. Consequently, we may substitute $\mathsf{gcd}(\mathsf{s}(\mathsf{s}(\mathsf{0})),\mathsf{s}(\mathsf{0}))$ for both $x$ and $y$ to satisfy the conditions of the CCPs.

It seems that there is still a lot of room for improvement with regard to infeasibility checking. So far all of the infeasibility methods employed by ConCon are unconditional and only approximate the conditional rewrite relation. Maybe we could overcome this problem by employing some external tool that takes conditions into account. Since our tree automata techniques are quite successful in showing infeasibility, the tree automata completion tool Timbuk [7, 9], which implements conditional tree automata completion, would be a natural candidate.

Another method that could be extended to allow conditions is the symbol transition graph. This was investigated further by Sternagel and Yamada [35], but was not yet formalized as part of IsaFoR.

In general, it would make sense to design, implement and (hopefully) verify a dedicated reachability tool for (conditional) term rewriting. The decomposition of reachability problems from Section 7.3 gives a nice modular framework for that. Not only ConCon would benefit from such a tool.

*Other Flavors.* For the time being CeTA only supports oriented CTRSs. In principle it should be possible to extend most of the presented methods for join CTRSs. Moreover, when employing conditional linearization [42] in order to show the unique normal form property with respect to conversions for non-left-linear TRSs, methods for semi-equational CTRSs are needed and could be formalized in future releases.

*Certification.* The main reason for the gap of two systems between the problems that ConCon can show confluent and the ones that CeTA can certify is due to the inductive symbol transition graph by Sternagel and Yamada [35].

## 10.3 Related Work

With the sole exception of the formalized unraveling result by Winkler and Thiemann [43], we are not aware of any other attempt to formally verify techniques for proving confluence of conditional term rewrite systems.

Concerning automated tools for proving CTRS confluence, ConCon is in good company: ACP [2], CO3[16] [26], CoScart[17] [12]. Among these tools ConCon is the only one that produces proof certificates.

## References

1. Antoy, S., Hanus, M.: Functional logic programming. Commun. ACM **53**(4), 74–85 (2010). doi:10.1145/1721654.1721675
2. Aoto, T., Yamaguchi, M.: ACP: System description for CoCo 2019. In: Proceedings of the 8th International Workshop on Confluence, p. 51 (2019)
3. Avenhaus, J., Loría-Sáenz, C.: On conditional rewrite systems with extra variables and deterministic logic programs. In: Proceedings of the 5th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, *Lecture Notes in Computer Science*, vol. 822, pp. 215–229. Springer (1994). doi:10.1007/3-540-58216-9_40
4. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
5. Bergstra, J., Klop, J.: Conditional rewrite rules: Confluence and termination. Journal of Computer and System Sciences **32**(3), 323–362 (1986). doi:10.1016/0022-0000(86)90033-4
6. Dershowitz, N., Okada, M., Sivakumar, G.: Confluence of conditional rewrite systems. In: Proceedings of the 1st International Workshop on Conditional and Typed Rewriting Systems, *Lecture Notes in Computer Science*, vol. 308, pp. 31–44. Springer (1988). doi:10.1007/3-540-19242-5_3
7. Feuillade, G., Genet, T.: Reachability in conditional term rewriting systems. In: Proceedings of the 4th International Workshop on First-Order Theorem Proving, *Electronic Notes in Theoretical Computer Science*, vol. 86, pp. 133–146 (2003). doi:10.1016/S1571-0661(04)80658-3
8. Genet, T.: Decidable approximations of sets of descendants and sets of normal forms. In: Proceedings of the 9th International Conference on Rewriting Techniques and Applications, *Lecture Notes in Computer Science*, vol. 1379, pp. 151–165. Springer (1998). doi:10.1007/BFb0052368
9. Genet, T., Viet Triem Tong, V.: Reachability analysis of term rewriting systems with timbuk. In: Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, *Lecture Notes in Computer Science*, vol. 2250, pp. 695–706. Springer (2001). doi:10.1007/3-540-45653-8_48

[16] https://www.trs.css.i.nagoya-u.ac.jp/co3/
[17] https://github.com/searles/RewriteTool/

10. Giesl, J., Rubio, A., Sternagel, C., Waldmann, J., Yamada, A.: The termination and complexity competition. In: Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), *Lecture Notes in Computer Science*, vol. 11429, pp. 156–166. Springer (2019). doi:10.1007/978-3-030-17502-3_10
11. Giesl, J., Thiemann, R., Schneider-Kamp, P.: Proving and disproving termination of higher-order functions. In: Proceedings of the 5th International Workshop on Frontiers of Combining Systems, *Lecture Notes in Computer Science*, vol. 3717, pp. 216–231. Springer (2005). doi:10.1007/11559306_12
12. Gmeiner, K.: CoScart: Confluence prover in Scala. In: Proceedings of the 5th International Workshop on Confluence, p. 82 (2016)
13. Gmeiner, K., Gramlich, B., Schernhammer, F.: On (Un)Soundness of Unravelings. In: Proceedings of the 21st International Conference on Rewriting Techniques and Applications, *Leibniz International Proceedings in Informatics*, vol. 6, pp. 119–134. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2010). doi:10.4230/LIPIcs.RTA.2010.119
14. Gmeiner, K., Nishida, N., Gramlich, B.: Proving confluence of conditional term rewriting systems via unravelings. In: Proceedings of the 2nd International Workshop on Confluence, pp. 35–39 (2013)
15. Haftmann, F., Nipkow, T.: Code generation via higher-order rewrite systems. In: M. Blume, N. Kobayashi, G. Vidal (eds.) Proceedings of the 10th International Symposium on Functional and Logic Programming (FLOPS), *Lecture Notes in Computer Science*, vol. 6009, pp. 103–117. Springer (2010). doi:10.1007/978-3-642-12251-4_9
16. Hanus, M.: On extra variables in (equational) logic programming. In: Proceedings of the 12th International Conference on Logic Programming, pp. 665–679. MIT Press (1995)
17. Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems. J. ACM **27**(4), 797–821 (1980)
18. Ida, T., Okui, S.: Outside-in conditional narrowing. IEICE Transactions on Information and Systems **E77-D**(6), 631–641 (1994)
19. Jacquemard, F.: Decidable approximations of term rewriting systems. In: Proceedings of the 7th International Conference on Rewriting Techniques and Applications, *Lecture Notes in Computer Science*, vol. 1103, pp. 362–376. Springer (1996). doi:10.1007/3-540-61464-8_65
20. Marlow, S.: Haskell 2010 language report. https://www.haskell.org/definition/haskell2010.pdf
21. Middeldorp, A.: Approximating dependency graphs using tree automata techniques. In: Proceedings of the 1st International Joint Conference on Automated Reasoning, *Lecture Notes in Artificial Intelligence*, vol. 2083, pp. 593–610 (2001). doi:10.1007/3-540-45744-5_49
22. Middeldorp, A., Hamoen, E.: Completeness Results for Basic Narrowing. Applicable Algebra in Engineering, Communication and Computing **5**, 213–253 (1994). doi:10.1007/BF01190830
23. Middeldorp, A., Nagele, J., Shintani, K.: Confluence competition 2019. In: Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), *Lecture Notes in Computer Science*, vol. 11429, pp. 25–40. Springer (2019). doi:10.1007/978-3-030-17502-3_2
24. Newman, M.: On theories with a combinatorial definition of equivalence. Annals of Mathematics **43**(2), 223–243 (1942). doi:10.2307/1968867
25. Nipkow, T.: Equational reasoning in Isabelle. Sci. Comput. Program. **12**(2), 123–149 (1989). doi:10.1016/0167-6423(89)90038-5
26. Nishida, N., Maeda, Y.: CO3 (version 2.0). In: Proceedings of the 8th International Workshop on Confluence, p. 50 (2019)
27. Nishida, N., Sakai, M., Sakabe, T.: Soundness of unravelings for conditional term rewriting systems via ultra-properties related to linearity. Logical Methods in Computer Science **8**, 1–49 (2012). doi:10.2168/LMCS-8(3:4)2012
28. Ohlebusch, E.: Advanced Topics in Term Rewriting. Springer (2002)
29. Sternagel, C., Sternagel, T.: Level-confluence of 3-CTRSs in Isabelle/HOL. In: Proceedings of the 4th International Workshop on Confluence, pp. 28–32 (2015). arXiv:1602.07115
30. Sternagel, C., Sternagel, T.: Certifying Confluence of Almost Orthogonal CTRSs via Exact Tree Automata Completion. In: Proceedings of the 1st International Conference on Formal Structures for Computation and Deduction, *Leibniz International Proceedings in Informatics*, vol. 51, pp. 29:1–29:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2016). doi:10.4230/LIPIcs.FSCD.2016.29

31. Sternagel, C., Sternagel, T.: Certifying Confluence of Quasi-Decreasing Strongly Deterministic Conditional Term Rewrite Systems. In: Proceedings of the 26th International Conference on Automated Deduction, *Lecture Notes in Computer Science*, vol. 10395, pp. 413–431. Springer (2017). doi:10.1007/978-3-319-63046-5_26
32. Sternagel, C., Thiemann, R.: Signature extensions preserve termination - an alternative proof via dependency pairs. In: Proceedings of the 19h International Workshop on Computer Science Logic, *Lecture Notes in Computer Science*, vol. 6247, pp. 514–528. Springer (2010). doi:10.1007/978-3-642-15205-4_39
33. Sternagel, C., Thiemann, R.: The Certification Problem Format. In: Proceedings of the 11th Workshop on User Interfaces for Theorem Provers, pp. 61–72 (2014). doi:10.4204/EPTCS.167.8
34. Sternagel, C., Winkler, S.: Certified equational reasoning via ordered completion. In: Proceedings of the 27th International Conference on Automated Deduction, Lecture Notes in Computer Science. Springer (2019). To appear
35. Sternagel, C., Yamada, A.: Reachability analysis for termination and confluence of rewriting. In: Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), *Lecture Notes in Computer Science*, vol. 11427, pp. 262–278. Springer (2019). doi:10.1007/978-3-030-17462-0_15
36. Sternagel, T., Middeldorp, A.: Conditional confluence (system description). In: Proceedings of the Joint 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications, *Lecture Notes in Computer Science*, vol. 8560, pp. 456–465. Springer (2014). doi:10.1007/978-3-319-08918-8_31
37. Sternagel, T., Middeldorp, A.: Infeasible conditional critical pairs. In: Proceedings of the 4th International Workshop on Confluence, pp. 13–17 (2015)
38. Sternagel, T., Sternagel, C.: Formalized confluence of quasi-decreasing, strongly deterministic conditional TRSs. In: Proceedings of the 5th International Workshop on Confluence, pp. 60–64 (2016). arXiv:1609.03341
39. Sternagel, T., Sternagel, C.: Certified non-confluence with ConCon 1.5. In: Proceedings of the 6th International Workshop on Confluence (2017). arXiv:1709.05162
40. Suzuki, T., Middeldorp, A., Ida, T.: Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In: Proceedings of the 6th International Conference on Rewriting Techniques and Applications, *Lecture Notes in Computer Science*, vol. 914, pp. 179–193. Springer (1995). doi:10.1007/3-540-59200-8_56
41. Thiemann, R., Sternagel, C.: Certification of termination proofs using CeTA. In: Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics, *Lecture Notes in Computer Science*, vol. 5674, pp. 452–468. Springer (2009). doi:10.1007/978-3-642-03359-9_31
42. de Vrijer, R.: Conditional linearization. Indagationes Mathematicae **10**(1), 145 – 159 (1999). doi:10.1016/S0019-3577(99)80012-3
43. Winkler, S., Thiemann, R.: Formalizing soundness and completeness of unravelings. In: Proceedings of the 10th International Workshop on Frontiers of Combining Systems, *Lecture Notes in Computer Science*, vol. 9322, pp. 239–255. Springer (2015). doi:10.1007/978-3-319-24246-0_15
44. Zankl, H., Felgenhauer, B., Middeldorp, A.: CSI – A confluence tool. In: Proceedings of the 23rd International Conference on Automated Deduction, *Lecture Notes in Artificial Intelligence*, vol. 6803, pp. 499–505. Springer (2011). doi:10.1007/978-3-642-22438-6_38