Conditional Confluence (System Description)*

Thomas Sternagel and Aart Middeldorp

University of Innsbruck, Innsbruck, Austria
{thomas.sternagel, aart.middeldorp}@uibk.ac.at

Abstract. This paper describes the *Con*ditional *Con*fluence tool, a fully automatic confluence checker for first-order conditional term rewrite systems. The tool implements various confluence criteria that have been proposed in the literature. A simple technique is presented to test conditional critical pairs for infeasibility, which makes conditional confluence criteria more useful. Detailed experimental data is presented.

Keywords: conditional term rewriting, confluence, automation.

1 Introduction

Confluence of term rewrite systems (TRSs) is an undecidable property. Nevertheless there are a number of tools [1, 10, 17] available to check for confluence of TRSs. For *conditional* TRSs (CTRSs) checking confluence is even harder and to date there was no automatic support. The Conditional Confluence tool—ConCon—aims to change this picture. The tool implements three different confluence criteria for oriented CTRSs that have been reported in the literature [2, 8, 16]. A simple technique for infeasibility of conditional critical pairs based on the tcap function is presented to (mildly) enhance the applicability of two of the confluence criteria.

The remainder of this paper is structured as follows. In Section 2 we sum up some basic facts about (conditional) rewriting the reader should be familiar with and we recall two transformations that are used to test for effective termination and confluence. The three implemented confluence criteria are described in Section 3. Section 4 is about infeasibility and contains a larger example. The tool is described in Section 5. A number of experiments have been conducted with ConCon. They are presented in Section 6. The paper concludes with some remarks on implementation issues, thoughts on extensions, and future work in Section 7.

2 Preliminaries

We assume knowledge of the basic notions regarding CTRSs (cf. [3, 15]). Let $\mathcal{R}$ be a CTRS. Let $\ell_1 \to r_1 \Leftarrow c_1$ and $\ell_2 \to r_2 \Leftarrow c_2$ be variants of rewrite rules

* The research described in this paper is supported by FWF (Austrian Science Fund) projects P22467 and I963.

of $\mathcal{R}$ without common variables and let $p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(\ell_2)$ such that ℓ_1 and $\ell_2|_p$ are unifiable. Let σ be a most general unifier of ℓ_1 and $\ell_2|_p$. If $\ell_1 \to r_1 \Leftarrow c_1$ and $\ell_2 \to r_2 \Leftarrow c_2$ are not variants or $p \neq \epsilon$ then the conditional equation $\ell_2\sigma[r_1\sigma]_p \approx \ell_2\sigma \Leftarrow c_1\sigma, c_2\sigma$ is called a *conditional critical pair* of $\mathcal{R}$. A conditional critical pair $s \approx t \Leftarrow c$ of a CTRS $\mathcal{R}$ is *joinable* if $s\sigma \downarrow_{\mathcal{R}} t\sigma$ for every substitution σ that satisfies c. Since in this paper we are concerned with *oriented* CTRSs, the latter means that $u\sigma \to^*_{\mathcal{R}} v\sigma$ for every equation $u \approx v$ in c. We say that $s \approx t \Leftarrow c$ is *infeasible* if there exists no substitution σ that satisfies c. The TRS obtained from a CTRS $\mathcal{R}$ by dropping the conditional parts of the rewrite rules is denoted by $\mathcal{R}_{\mathsf{u}}$. We say that $\mathcal{R}$ is *normal* if every right-hand side of every condition in every rule is a ground normal form with respect to $\mathcal{R}_{\mathsf{u}}$. Rewrite rules $\ell \to r \Leftarrow c$ of CTRSs are classified according to the distribution of variables among ℓ, r, and c, as follows:

type	requirement
1	$\mathcal{V}\mathsf{ar}(r) \cup \mathcal{V}\mathsf{ar}(c) \subseteq \mathcal{V}\mathsf{ar}(\ell)$
2	$\mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(\ell)$

type	requirement
3	$\mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(\ell) \cup \mathcal{V}\mathsf{ar}(c)$
4	no restrictions

An n-CTRS contains only rules of type n. So a 1-CTRS contains no extra variables, a 2-CTRS may only contain extra variables in the conditions, and a 3-CTRS may also have extra variables in the right-hand sides provided these occur in the corresponding conditional part. The set of variables occurring in a sequence of terms $t_1, \ldots, t_n$ is denoted by $\mathcal{V}\mathsf{ar}(t_1, \ldots, t_n)$. Likewise the function $\mathsf{var}(t_1, \ldots, t_n)$ returns the elements of $\mathcal{V}\mathsf{ar}(t_1, \ldots, t_n)$ in an arbitrary but fixed order. An oriented CTRS $\mathcal{R}$ is called *deterministic* if for every rule $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$ in $\mathcal{R}$ we have $\mathcal{V}\mathsf{ar}(s_i) \subseteq \mathcal{V}\mathsf{ar}(\ell, t_1, \ldots, t_{i-1})$ with $1 \leqslant i \leqslant n$.

An oriented CTRS $\mathcal{R}$ is *quasi-decreasing* if there exists a well-founded order $>$ with the subterm property that extends $\to_{\mathcal{R}}$ such that $\ell\sigma > s_i\sigma$ for all $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n \in \mathcal{R}$, $1 \leqslant i \leqslant n$, and substitutions σ with $s_j\sigma \to^*_{\mathcal{R}} t_j\sigma$ for $1 \leqslant j < i$. Quasi-decreasingness ensures termination and, for finite CTRSs, computability of the rewrite relation. We recall two transformations from deterministic 3-CTRSs to TRSs that can be used to show quasi-decreasingness.

Unravelings were first introduced in [13]. Unravelings split conditional rules into several unconditional rules and the conditions are encoded using new function symbols. Originally they were used to study the correspondence between properties of CTRSs and TRSs as well as modularity of CTRSs. The unraveling defined below goes back to [12]. We use the formulation in [15, p. 212]. It simulates the conditional rules from a CTRS $\mathcal{R}$ by a sequence of applications of rules from the TRS $\mathbb{U}(\mathcal{R})$, in effect verifying the conditions from left to right until all the conditions are satisfied and the last rule yielding the original right-hand side may be applied.

Definition 1. *Every deterministic 3-CTRS $\mathcal{R}$ is mapped to the TRS $\mathbb{U}(\mathcal{R})$ obtained from $\mathcal{R}$ by replacing every conditional rule $\rho\colon \ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx$*

t_n *with* $n \geqslant 1$ *in* $\mathcal{R}$ *with*

$$\begin{aligned} \ell &\to U^1_\rho(s_1, \mathsf{var}(\ell)) \\ U^1_\rho(t_1, \mathsf{var}(\ell)) &\to U^2_\rho(s_2, \mathsf{var}(\ell, t_1)) \\ &\dots \\ U^n_\rho(t_n, \mathsf{var}(\ell, t_1, \dots, t_{n-1})) &\to r \end{aligned}$$

where U^i_ρ *are fresh function symbols.*

In our implementation we use the variant of $\mathbb{U}$ sketched in [15, Example 7.2.49] and formalized in [8, Definition 6]. In this variant certain U-symbols originating from different rewrite rules are shared, in order to reduce the number of critical pairs and thereby increasing the chances of obtaining a confluent TRS.

The second transformation, introduced in [2], from deterministic 3-CTRSs to TRSs does not use any additional symbols and it does not aim to simulate rewriting in the CTRS. Hence its use is limited to show quasi-decreasingness.

Definition 2. *Every deterministic 3-CTRS* $\mathcal{R}$ *is mapped to the TRS* $\mathbb{V}(\mathcal{R})$ *obtained from* $\mathcal{R}$ *by replacing every conditional rule* $\ell \to r \Leftarrow s_1 \approx t_1, \dots, s_n \approx t_n$ *with* $n \geqslant 1$ *in* $\mathcal{R}$ *with*

$$\ell \to s_1\sigma_0 \qquad \dots \qquad \ell \to s_n\sigma_{n-1} \qquad \ell \to r\sigma_n$$

for the substitutions $\sigma_0, \dots, \sigma_n$ *inductively defined as follows:*

$$\sigma_i = \begin{cases} \varepsilon & \text{if } i = 0 \\ \sigma_{i-1} \cup \{x \mapsto s_i\sigma_{i-1} \mid x \in \mathcal{V}\mathsf{ar}(t_i) \setminus \mathcal{V}\mathsf{ar}(\ell, t_1, \dots, t_{i-1})\} & \text{if } 0 < i \leqslant n \end{cases}$$

The following lemma shows how the transformations are used to obtain quasi-decreasingness. The first condition is from [15, p. 214] while the second one is a combination of [15, Lemma 7.2.6] and [15, Proposition 7.2.68]. It is unknown whether the first condition is implied by the second (cf. [15, p. 229]).

Lemma 3. *A deterministic 3-CTRS* $\mathcal{R}$ *is quasi-decreasing if* $\mathbb{U}(\mathcal{R})$ *is terminating or* $\mathbb{V}(\mathcal{R})$ *is simply terminating.* □

3 Three Confluence Criteria

Our tool implements three known confluence criteria [2, 8, 16]. The first criterion is from Avenhaus and Loría-Sáenz [2, Theorem 4.1]. Its applicability is restricted to quasi-decreasing and *strongly irreducible* deterministic 3-CTRSs. A term t is called *strongly irreducible* if $t\sigma$ is a normal form for every normalized[1] substitution σ. We say that $\mathcal{R}$ is strongly irreducible if the right-hand side of every condition in every conditional rewrite rule is strongly irreducible. Strong irreducibility is undecidable. In our tool we use the following decidable approximation [2]: no non-variable subterm of a right-hand side of a condition unifies with the left-hand side of a rule (after renaming).

[1] A normalized substitution maps variables to normal forms.

Theorem A *A quasi-decreasing strongly irreducible deterministic 3-CTRS $\mathcal{R}$ is confluent if and only if all critical pairs of $\mathcal{R}$ are joinable.* □

The second confluence criterion is from Suzuki *et al.* [16, Section 7]. It does not impose any termination assumption, but forbids (feasible) critical pairs and requires the properties defined below, which are obviously computable. A CTRS $\mathcal{R}$ is *right-stable* if every rewrite rule $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$ in $\mathcal{R}$ satisfies $\mathcal{V}\mathsf{ar}(\ell, s_1, \ldots, s_i, t_1, \ldots, t_{i-1}) \cap \mathcal{V}\mathsf{ar}(t_i) = \varnothing$ and t_i is either a linear constructor term or a ground $\mathcal{R}_{\mathsf{u}}$-normal form, for all $1 \leqslant i \leqslant n$. An oriented CTRS $\mathcal{R}$ is *properly oriented* if for every rewrite rule $\ell \to r \Leftarrow c$ with $\mathcal{V}\mathsf{ar}(r) \not\subseteq \mathcal{V}\mathsf{ar}(\ell)$ in $\mathcal{R}$ the conditional part c can be written as $s_1 \approx t_1, \ldots, s_m \approx t_m, s'_1 \approx t'_1, \ldots, s'_n \approx t'_n$ such that the following two conditions are satisfied: $\mathcal{V}\mathsf{ar}(s_i) \subseteq \mathcal{V}\mathsf{ar}(\ell, t_1, \ldots, t_{i-1})$ for all $1 \leqslant i \leqslant m$ and $\mathcal{V}\mathsf{ar}(r) \cap \mathcal{V}\mathsf{ar}(s'_i, t'_i) \subseteq \mathcal{V}\mathsf{ar}(\ell, t_1, \ldots, t_m)$ for all $1 \leqslant i \leqslant n$.

Theorem B *Almost orthogonal properly oriented right-stable 3-CTRSs are confluent.* □

The third criterion is a recent result by Gmeiner *et al.* [8, Theorem 9]. It employs the notion of *weak left-linearity*, which is satisfied for a deterministic CTRS $\mathcal{R}$ if $x \notin \mathcal{V}\mathsf{ar}(r, s_1, \ldots, s_n)$ for every rule $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$ in $\mathcal{R}$ and variable x that appears more than once in $\ell, t_1, \ldots, t_n$.

Theorem C *A weakly left-linear deterministic CTRS $\mathcal{R}$ is confluent if $\mathbb{U}(\mathcal{R})$ is confluent.* □

Here the modified version of the unraveling $\mathbb{U}$ described after Definition 1 is used. The three criteria are pairwise incompatible, as shown in the following examples.

Example 4. The oriented 1-CTRS $\mathcal{R}$ consisting of the following four rules

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

is weakly left-linear and deterministic and its unraveling $\mathbb{U}(\mathcal{R})$

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

is confluent, hence $\mathcal{R}$ is confluent by Theorem C. Since $\mathcal{R}$ is neither left-linear nor strongly irreducible, Theorems A and B are not applicable.

Example 5. The normal 2-CTRS consisting of the rule

$$\mathsf{h}(x) \to \mathsf{g}(x) \Leftarrow \mathsf{f}(x, y) \approx \mathsf{b}$$

is orthogonal, properly oriented, and right-stable and therefore confluent by Theorem B but not deterministic so Theorems A and C do not apply. Hence this example contradicts the claim in [8] that Theorem B is a corollary of Theorem C.

Example 6. The normal 1-CTRS consisting of the rule

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

is quasi-decreasing, strongly irreducible, and non-overlapping, and thus confluent by Theorem A. Since the system is not (weakly) left-linear, Theorems B and C do not apply.

4 Infeasibility

The applicability of Theorems A and B strongly depends on the presence of critical pairs. Many natural examples employ rules which only yield a couple of critical pairs which are in fact all infeasible.

Infeasibility is undecidable in general. Two sufficient conditions are described in the literature: [9, Appendix A] and [2, Definition 4.4]. The former method can only be used in very special cases (left-linear constructor-based join systems without extra variables and using "strict" semantics). The use of the latter method is restricted to quasi-decreasing strongly irreducible deterministic 3-CTRSs (like in Theorem A) and is described below.

Given a critical pair $s \approx t \Leftarrow c$, the conditions in c are transformed into a TRS $\mathcal{C} = \{\overline{u} \to \overline{v} \mid u \approx v \in c\}$ where $\overline{t}$ is the result of replacing every $x \in \mathcal{V}\mathsf{ar}(c)$ occurring in t by a fresh constant c_x. If there is a left-hand side u in c such that $\overline{u}_2 \; {}^{*}_{\mathcal{R}\cup\mathcal{C}}\!\leftarrow \overline{u} \to^{*}_{\mathcal{R}\cup\mathcal{C}} \overline{u}_1$ and u_1, u_2 are strongly irreducible and not unifiable then $s \approx t \Leftarrow c$ is infeasible. The same method can also be used as sufficient condition for joinability [2]: $s \approx t \Leftarrow c$ is joinable if $\overline{s} \to^{*}_{\mathcal{R}\cup\mathcal{C}} \cdot \; {}^{*}_{\mathcal{R}\cup\mathcal{C}}\!\leftarrow \overline{t}$.

Our new technique for infeasibility is based on the tcap function, which was introduced to obtain a better approximation of dependency graphs [7] and later used as a sufficient check for non-confluence for TRSs [17]. It is defined as follows. If t is a variable then $\mathsf{tcap}(t)$ is a fresh variable and if $t = f(t_1, \ldots, t_n)$ then we let $u = f(\mathsf{tcap}(t_1), \ldots, \mathsf{tcap}(t_n))$ and define $\mathsf{tcap}(t)$ to be u if u does not unify with the left-hand side of a rule in $\mathcal{R}$, and a fresh variable otherwise.

Lemma 7. *Let $\mathcal{R}$ be an oriented CTRS. A conditional critical pair $s \approx t \Leftarrow c$ of $\mathcal{R}$ is infeasible if there exists an equation $u \approx v \in c$ such that $\mathsf{tcap}(u)$ does not unify with v.* □

We conclude this section with an example illustrating that Theorem A benefits from the new infeasibility criterion of Lemma 7.

Example 8. Consider the following CTRS $\mathcal{R}_{\mathsf{min}}$ from [11]:

$$
\begin{array}{rlll}
0 < \mathsf{s}(x) \to \mathsf{true} & \mathsf{min}(x : \mathsf{nil}) \to x & & (1)\\
x < 0 \to \mathsf{false} & \mathsf{min}(x : xs) \to x & \Leftarrow\ x < \mathsf{min}(xs) \approx \mathsf{true} & (2)\\
\mathsf{s}(x) < \mathsf{s}(y) \to x < y & \mathsf{min}(x : xs) \to \mathsf{min}(xs) & \Leftarrow\ x < \mathsf{min}(xs) \approx \mathsf{false} & (3)\\
 & \mathsf{min}(x : xs) \to \mathsf{min}(xs) & \Leftarrow\ \mathsf{min}(xs) \approx x & (4)
\end{array}
$$

To check whether Theorem C is able to show confluence of $\mathcal{R}_{\mathsf{min}}$ we look at the result of the optimized version of the unraveling $\mathbb{U}$ from Definition 1 on rules (2) to (4):

$$
\begin{array}{ll}
\mathsf{min}(x : xs) \to \mathsf{U}_1(x < \mathsf{min}(xs), x, xs) & \mathsf{min}(x : xs) \to \mathsf{U}_2(\mathsf{min}(xs), x, xs)\\
\mathsf{U}_1(\mathsf{true}, x, xs) \to x & \mathsf{U}_2(x, x, xs) \to \mathsf{min}(xs)\\
\mathsf{U}_1(\mathsf{false}, x, xs) \to \mathsf{min}(xs) &
\end{array}
$$

There is a peak $\mathsf{U}_1(x < \mathsf{min}(xs), x, xs) \leftarrow \mathsf{min}(x : xs) \rightarrow \mathsf{U}_2(\mathsf{min}(xs), x, xs)$ between different normal forms of $\mathbb{U}(\mathcal{R}_{\mathsf{min}})$ and hence $\mathbb{U}(\mathcal{R}_{\mathsf{min}})$ is non-confluent. So Theorem C cannot show confluence of $\mathcal{R}_{\mathsf{min}}$. Theorem B does not apply here because $\mathcal{R}_{\mathsf{min}}$ is not right-stable. For Theorem A we compute critical pairs. There are twelve but for symmetry reasons we only have to consider six of them:

$$\begin{aligned}
x &\approx x && \Leftarrow\ x < \mathsf{min}(\mathsf{nil}) \approx \mathsf{true} && (1,2)\\
\mathsf{min}(\mathsf{nil}) &\approx x && \Leftarrow\ x < \mathsf{min}(\mathsf{nil}) \approx \mathsf{false} && (1,3)\\
\mathsf{min}(\mathsf{nil}) &\approx x && \Leftarrow\ \mathsf{min}(\mathsf{nil}) \approx x && (1,4)\\
\mathsf{min}(xs) &\approx x && \Leftarrow\ x < \mathsf{min}(xs) \approx \mathsf{true},\ x < \mathsf{min}(xs) \approx \mathsf{false} && (2,3)\\
\mathsf{min}(xs) &\approx x && \Leftarrow\ x < \mathsf{min}(xs) \approx \mathsf{true},\ \mathsf{min}(xs) \approx x && (2,4)\\
\mathsf{min}(xs) &\approx \mathsf{min}(xs) && \Leftarrow\ x < \mathsf{min}(xs) \approx \mathsf{false},\ \mathsf{min}(xs) \approx x && (3,4)
\end{aligned}$$

The pairs (1,2) and (3,4) are trivial. The terms $\mathsf{tcap}(x < \mathsf{min}(\mathsf{nil})) = x' < \mathsf{min}(\mathsf{nil})$ and false are not unifiable, hence (1,3) is infeasible by Lemma 7. The critical pair (2,3) can be shown to be infeasible by the method described at the top of page 5 (as well as by Lemma 7) and (1,4) and (2,4) can be shown to be joinable by the same method. So Theorem A applies and we conclude that $\mathcal{R}_{\mathsf{min}}$ is confluent.

5 Design and Implementation

ConCon is written in Scala 2.10, an object-functional programming language. Scala compiles to Java byte code and therefore is easily portable to different platforms. ConCon is available under the LGPL license and may be downloaded from:

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

In order to use the full power of ConCon one needs to have some termination checker understanding the TPDB[2] format and some confluence checker understanding the same format installed on one's system. One may have to adjust the paths and flags of these programs in the file `concon.ini`, which should reside in the same directory as the `concon` executable. For input we support the XML[3] format as well as a modified version of the TRS format of the TPDB.[4]

The modification concerns a new declaration `CONDITIONTYPE`, which may be set to `SEMI-EQUATIONAL`, `JOIN`, or `ORIENTED`. Although for now ConCon works on oriented CTRSs we designed the `CONDITIONTYPE` to anticipate future developments. In the conditional part of the rules we only allow `==` as relation, since the exact interpretation is inferred from the `CONDITIONTYPE` declaration:

```
(CONDITIONTYPE ORIENTED)
(VAR x)
```

[2] http://www.lri.fr/~marche/tpdb/format.html
[3] http://www.termination-portal.org/wiki/XTC_Format_Specification
[4] http://termination-portal.org/wiki/TPDB

```
(RULES
  not(x) -> false | x == true
  not(x) -> true  | x == false
)
```

This modified TRS format is closer to the newer XML version and makes it very easy to interpret, say, a given join CTRS as an oriented CTRS (by just modifying the `CONDITIONTYPE`).

ConCon is operated through a command line interface described below. In addition to the command line version there is also an easy to use web interface available on the ConCon website.

Usage Just starting the tool without any options or input file as follows

```
java -jar concon_2.10-1.1.0.0.min.jar
```

will output a short usage description. We will abbreviate this command by `./concon` in the following. The flag `--conf` may be used to configure the employed confluence criteria. The flag takes a list of criteria which are tried in the given order. If a method is successful the rest of the list is skipped. By default ConCon uses all the available confluence criteria in the following order:[5]

- `U` Check whether the input system is unconditional, if so give it to an external unconditional confluence checker.
- `B` Try Theorem B.
- `C` Try Theorem C using an external unconditional confluence checker.
- `A` Try Theorem A using an external termination checker.

One may always add a timeout at the end of ConCon's parameter list. The default timeout is 60 seconds. When calling ConCon with an input file like

```
./concon 292.trs
```

it will just try to apply all confluence criteria in sequence with the default timeout as explained above. The first line of the output will be one of `YES`, `NO`, or `MAYBE`, followed by the input system, and finally a textual description of how ConCon did conclude the given answer. One may use `-a`, `-s`, and `-p` to prevent output of the answer, the input system, and the textual description, respectively.

If one is only interested in the critical pairs of the system and which of them can be shown to be infeasible, one may use the following call

```
./concon -c 292.trs
```

the `-c` causes ConCon to print all overlaps and the associated (conditional) critical pairs of the system, and indicates whether they could be shown infeasible.

[5] Theorem B does not need calls to external programs and in our experiments Theorem C produced an answer faster than Theorem A on average.

In order to check the input system for quasi-decreasingness the flag `-q` may be used. In addition one may use the option `--ter` together with one of the strings `u` or `v` to restrict the transformation to use for the termination check. This method gives the transformed unconditional system to an external termination checker.

The flag `-t` may be used to tell ConCon to just apply a transformation and output the result. The flag takes a string parameter specifying which transformation to use. The available options are `u`, `uopt` and `v` standing for the transformations of Definition 1, its modified version, and Definition 2, respectively.

Many syntactic criteria for CTRSs, like proper-orientedness or weak left-linearity, are tedious to check by hand. Other properties of interest, like quasi-decreasingness, are undecidable. Executing the call

```
./concon -l 292.trs
```

results in a list of properties of the input CTRS.

6 Experiments

We have collected a number of examples from the literature. Our collection currently consists of 129 CTRSs, including the 3 new examples in Section 3, which we extracted from 32 different sources. Of these 129 CTRSs, 101 are presented as oriented CTRSs in the literature. The corresponding files in the modified TPDB format can be downloaded from the ConCon website. Additionally they have been added to the confluence problems database.[6] This collection should also be of interest for the termination competition[7] since the CTRS category of TPDB contains a mere 7 examples.

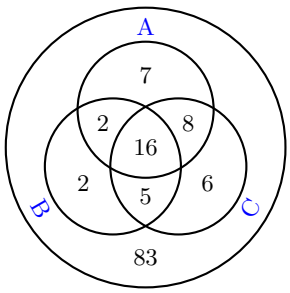
The experiments we describe here were carried out on a 64bit GNU/Linux machine with an Intel® Core™ i7-3520M processor clocked at 2.90GHz and 8GB of RAM using the tool `parallel`.[8] The kernel version is `3.14.1-1-ARCH`. The version of Java on this machine is 1.7.0_55. We had to increase the stack size to 20MB using the JVM flag `-Xss20M` to prevent stack overflows caused by parsing deep terms like in the file `313.trs`. The following external tools were used in the experiments:

- CSI, version 0.4 (call: `csi - trs 30`)[9]
- $\mathsf{T_TT_2}$, version 1.16 (call: `ttt2 - trs 30`)[10]

First we checked confluence of the given systems. The timeout was set to one minute.

Figure 1a gives an overview of how many systems could be shown to be confluent by which of the three theorems. More details for 6 of the CTRSs are listed

[6] http://coco.nue.riec.tohoku.ac.jp/problems/
[7] http://termcomp.uibk.ac.at/
[8] http://www.gnu.org/s/parallel
[9] http://cl-informatik.uibk.ac.at/software/csi/
[10] http://cl-informatik.uibk.ac.at/software/ttt2/



(a) 129 CTRSs

CTRS	source	theorem A	B	C	CCPs #	=	⋈	↓
`264.trs`	6	0.5	–	–	0	–	–	–
`286.trs`	4	–	–	2.2	0	–	–	–
`287.trs`	5	–	0.4	–	0	–	–	–
`292.trs`	8	0.6	–	×	12	4	6	2
`324.trs`	[14]	–	–	×	4	0	0	0
`336.trs`	[16]	–	0.5	–	2	0	2	0

(b) 6 selected CTRSs

Fig. 1: Confluence results.

in Figure 1b. The columns 'A' to 'C' list the time in seconds in case of success, '–' if the theorem was not applicable because of the syntactic preconditions, and '×' if the method will never be able to show confluence for the system. The first 3 CTRSs can only be shown to be confluent by one of the theorems. The next one (`292.trs`) is Example 8. Example `324.trs` cannot be shown confluent by any of the implemented methods. As can be seen in Figure 1a, this actually holds for the majority of the 129 CTRSs. Example `336.trs` requires the method of Lemma 7 to show its conditional critical pairs infeasible, afterwards Theorem B is applicable. In the last four columns we list for each of the 6 CTRSs the number of conditional critical pairs, and whether they are trivial (column '='), or could be shown to be infeasible (column '⋈') or joinable (column '↓').

The ConCon website contains more detailed experimental results, including a comparison of the two sufficient conditions in Lemma 3 for quasi-decreasingness.

7 Concluding Remarks

We presented a tool which implements three different methods to show confluence of oriented CTRSs. A simple sufficient criterion for infeasibility increased the applicability of two of the methods. This is clearly a first step and our experiments show that there is lots of room for improvements:

- Several of the systems in our test bed are in fact non-confluent, so methods to prove non-confluence of CTRSs are in demand.
- Many of the systems have infeasible critical pairs but our current criterion for infeasibility is not powerful enough to show this. Our own investigations show that progress here is hard to achieve. For oriented CTRSs, infeasibility is a reachability problem and techniques based on tree automata completion are a natural candidate for investigation.
- Furthermore, so far we have no good support for conditional rewriting, which is needed to check joinability of (feasible) critical pairs. From early investigations by Ganzinger [6], Zhang and Rémy [18, 19], Avenhaus and Loría-

Sáenz [2], and others in we know that implementing conditional rewriting is a highly complex problem.

Finally, confluence methods for semi-equational [4] and in particular join CTRSs [5, 16] are well-investigated and should be implemented.

References

1. T. Aoto, J. Yoshida, and Y. Toyama. Proving confluence of term rewriting systems automatically. In *Proc. 20th RTA*, volume 5595 of *LNCS*, pages 93–102, 2009.
2. J. Avenhaus and C. Loría-Sáenz. On conditional rewrite systems with extra variables and deterministic logic programs. In *Proc. 5th LPAR*, volume 822 of *LNAI*, pages 215–229, 1994.
3. F. Baader and T. Nipkow. *Term Rewriting and All That*. CUP, 1998.
4. J.A. Bergstra and J.W. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986.
5. N. Dershowitz, M. Okada, and G. Sivakumar. Confluence of conditional rewrite systems. In *Proc. 1st CTRS*, volume 308 of *LNCS*, pages 31–44, 1988.
6. H. Ganzinger. A completion procedure for conditional equations. *Journal of Symbolic Computation*, 11(1/2):51–81, 1991.
7. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proc. 5th FRoCoS*, volume 3717 of *LNAI*, pages 216–231, 2005.
8. K. Gmeiner, N. Nishida, and B. Gramlich. Proving confluence of conditional term rewriting systems via unravelings. In *Proc. 2nd IWC*, pages 35–39, 2013.
9. J.C. González-Moreno, M.T. Hortalá-González, and M. Rodríguez-Artalejo. Denotational versus declarative semantics for functional programming. In *Proc. 5th CSL*, volume 626 of *LNCS*, pages 134–148, 1992.
10. N. Hirokawa and D. Klein. Saigawa: A confluence tool. In *Proc. 1st IWC*, page 49, 2012.
11. E. Kounalis and M. Rusinowitch. A proof system for conditional algebraic specifications. In *Proc. 2nd CTRS*, volume 516 of *LNCS*, pages 51–63, 1991.
12. M. Marchiori. On deterministic conditional rewriting. Computation Structures Group Memo 405, MIT Laboratory for Computer Science, 1987.
13. M. Marchiori. Unravelings and ultra-properties. In *Proc. 5th ALP*, volume 1139 of *LNCS*, pages 107–121, 1996.
14. N. Nishida, M. Sakai, and T. Sakabe. Soundness of unravelings for conditional term rewriting systems via ultra-properties related to linearity. *Logical Methods in Computer Science*, 8:1–49, 2012.
15. E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002.
16. T. Suzuki, A. Middeldorp, and T. Ida. Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In *Proc. 6th RTA*, volume 914 of *LNCS*, pages 179–193, 1995.
17. H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – A confluence tool. In *Proc. 23rd CADE*, volume 6803 of *LNAI*, pages 499–505, 2011.
18. H. Zhang. Implementing contextual rewriting. In *Proc. 3nd CTRS*, volume 656 of *LNCS*, pages 363–377, 1993.
19. H. Zhang and J.-L. Remy. Contextual rewriting. In *Proc. 1st RTA*, volume 202 of *LNCS*, pages 46–62, 1985.