Confluence Competition 2019

Aart Middeldorp[1](✉), Julian Nagele[2], and Kiraku Shintani[3]

[1] Department of Computer Science, University of Innsbruck, Innsbruck, Austria
aart.middeldorp@uibk.ac.at
[2] School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK
j.nagele@qmul.ac.uk
[3] School of Information Science, JAIST, Nomi, Japan
s1820017@jaist.ac.jp

Abstract. We report on the 2019 edition of the Confluence Competition, a competition of software tools that aim to prove or disprove confluence and related (undecidable) properties of rewrite systems automatically.

Keywords: Confluence · Term rewriting · Automation

1 Introduction

The Confluence Competition (CoCo)[1] is an annual competition of software tools that aim to prove or disprove confluence and related (undecidable) properties of a variety of rewrite formalisms automatically. Initiated in 2012, CoCo runs live in a single slot at a conference or workshop and is executed on the cross-community competition platform StarExec [1]. For each category, 100 suitable problems are randomly selected from the online database of confluence problems (COPS). Participating tools must answer YES or NO within 60 s, followed by a justification that is understandable by a human expert; any other output signals that the tool could not determine the status of the problem. CoCo 2019 features new categories on commutation, infeasibility problems, and confluence of string rewrite systems.

Confluence provides a general notion of determinism and has been conceived as one of the central properties of rewriting. A rewrite system $\mathcal{R}$ is a set of directed equations, so called rewrite rules, which induces a rewrite relation $\to_{\mathcal{R}}$ on terms. We provide a simple example.

Example 1. Consider the rewrite system $\mathcal{R}$ consisting of the rules

$$\begin{aligned} \mathsf{0} + y &\to y & \qquad \mathsf{0} \times y &\to y \\ \mathsf{s}(x) + y &\to \mathsf{s}(x + y) & \qquad \mathsf{s}(x) \times y &\to (x \times y) + y \end{aligned}$$

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

This research is supported by FWF (Austrian Science Fund) project P27528.

© The Author(s) 2019
D. Beyer et al. (Eds.): TACAS 2019, Part III, LNCS 11429, pp. 25–40, 2019.
https://doi.org/10.1007/978-3-030-17502-3_2

$$\begin{tikzcd}
\forall s\,\forall t\,\forall u & & s \arrow[dl, "*"', "\mathcal{R}"] \arrow[dr, "*", "\mathcal{R}"'] & \\
& t \arrow[dr, dashed, "*", "\mathcal{R}"'] & & u \arrow[dl, dashed, "*"', "\mathcal{R}"] \\
\exists v & & v &
\end{tikzcd}$$

Fig. 1. Confluence.

which can be viewed as a specification of addition and multiplication over natural numbers in unary notation. Computing $2 \times (1 + 2)$ amounts to evaluating the term $s = \mathsf{s}(\mathsf{s}(\mathsf{0})) \times (\mathsf{s}(\mathsf{0}) + \mathsf{s}(\mathsf{s}(\mathsf{0})))$. This is done by matching a subterm with the left-hand side of a rewrite rule, and if matching succeeds, replacing that subterm by the right-hand side of the rule after applying the matching substitution to its variables. For instance, the subterm $\mathsf{s}(\mathsf{0}) + \mathsf{s}(\mathsf{s}(\mathsf{0}))$ of s matches the left-hand side of the rule $\mathsf{s}(x) + y \to \mathsf{s}(x + y)$, with matching substitution $\{x \mapsto \mathsf{0}, y \mapsto \mathsf{s}(\mathsf{s}(\mathsf{0}))\}$. Hence the subterm can be replaced by $\mathsf{s}(\mathsf{0} + \mathsf{s}(\mathsf{s}(\mathsf{0})))$. It follows that s rewrites (in a single step) to the term $t = \mathsf{s}(\mathsf{s}(\mathsf{0})) \times \mathsf{s}(\mathsf{0} + \mathsf{s}(\mathsf{s}(\mathsf{0})))$. Continuing this process from t eventually results in the term $\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0}))))))$. This term cannot be simplified further. Such terms are called normal forms.

In the above example there are several ways to evaluate the term s. The choice does not matter since all maximal rewrite sequences terminate in the same normal form, which is readily checked. This property not only holds for the term s, but for all terms that can be constructed from the symbols in the rules. *Confluence* is the property that guarantees this. A rewrite system $\mathcal{R}$ is confluent if the inclusion ${}_{\mathcal{R}}^{*}\!\leftarrow \cdot \to_{\mathcal{R}}^{*} \subseteq \to_{\mathcal{R}}^{*} \cdot {}_{\mathcal{R}}^{*}\!\leftarrow$ holds. Here $\to_{\mathcal{R}}^{*}$ denotes the transitive reflexive closure of the one-step rewrite relation $\to_{\mathcal{R}}$, ${}_{\mathcal{R}}^{*}\!\leftarrow$ denotes the inverse of $\to_{\mathcal{R}}^{*}$, and $\cdot$ denotes relational composition. A more graphical definition of confluence is presented in Fig. 1. The precise notions of rewrite rules, associated rewrite steps, and terms to be rewritten vary from formalism to formalism.

2 Categories

In recent years the focus in confluence research has shifted towards the development of automatable techniques for confluence proofs. To stimulate these developments the Confluence Competition has been set up in 2012. Since its creation with 4 tools competing in 2 categories, CoCo has grown steadily and will feature the following 12 categories in 2019:

TRS/CPF-TRS The two original categories are about confluence of *first-order term rewriting.* CPF-TRS is a category for *certified* confluence proofs, where participating tools must generate certificates that are checked by an independent certifier.

CTRS/CPF-CTRS These two categories, introduced respectively in 2014 and 2015, are concerned (certified) confluence of *conditional term rewriting*, a formalism in which rewrite rules come equipped with conditions that are evaluated recursively using the rewrite relation.

HRS This category, introduced in 2015, deals with confluence of *higher-order rewriting*, i.e., rewriting with binders and functional variables.

GCR This category is about *ground* confluence of *many-sorted* term rewrite systems and was also introduced in 2015.

NFP/UNC/UNR These three categories, introduced in 2016, are about properties of first-order term rewrite systems related to unique normal forms, namely, *the normal form property* (NFP), *unique normal forms with respect to conversion* (UNC), and *unique normal forms with respect to reduction* (UNR).

COM This new category is about *commutation* of first-order rewrite systems.

INF This new category is about *infeasibility* problems.

SRS This new category is concerned with confluence of *string rewriting*.

The new categories are described in detail in Sect. 5. Descriptions of the other categories can be found in the CoCo 2015 [2] and 2018 [3] reports, and on the CoCo website (see Footnote 1). The underlying problem format is the topic of the next section.

3 Confluence Problems

Tools participating in CoCo are given problems from the database of *confluence problems* (COPS)[2] in a format suitable for the category in which the tools participate. Besides commutation and infeasibility problems, which are described in Sect. 5, four different formats are supported: TRS, CTRS, MSTRS, and HRS. As these formats were simplified recently, we present the official syntax below in four subsections.

In addition to the format, *tags* are used to determine suitable problems for CoCo categories. For instance, for the CTRS category, selected problems must have the `3_ctrs` and `oriented` tags. Such tags are automatically computed when problems are submitted to COPS. Detailed information on COPS, including a description of the tagging mechanism, can be found in [4].

3.1 TRS Format

The format for first-order rewrite systems comes in two versions: a basic version and an extended version. The latter contains an additional signature declaration which is used to define function symbols that do not appear in the rewrite rules.

[2] https://cops.uibk.ac.at/.

The basic format is a simplification of the old TPDB format,[3] according to the following grammar:

```
trs      ::= [(VAR idlist)] (RULES rulelist) [(COMMENT string)]
idlist   ::= ϵ | id idlist
rulelist ::= ϵ | rule rulelist
rule     ::= term -> term
term     ::= id | id() | id(termlist)
termlist ::= term | term, termlist
```

Here *string* is any sequence of characters and *id* is any nonempty sequence of characters not containing whitespace, the characters `(`, `)`, `"`, `,`, `|`, `\`, and the sequences `->`, `==`, `COMMENT`, `VAR`, and `RULES`. In `(VAR` *idlist*`)` the variables of the TRS are declared. If this is missing, the TRS is ground. Symbols (*id*) appearing in the `(RULES` *rulelist*`)` declaration that were not declared as variables are function symbols. If they appear multiple times, they must be used with the same number (arity) of arguments. Here is an example of the basic format, COPS #1:

```
(VAR x y)
(RULES
  f(x,y) -> x
  f(x,y) -> f(x,g(y))
  g(x) -> h(x)
  F(g(x),x) -> F(x,g(x))
  F(h(x),x) -> F(x,h(x))
)
(COMMENT
doi:10.1007/BFb0027006
[1] Example 6
submitted by: Takahito Aoto, Junichi Yoshida, and Yoshihito Toyama
)
```

In the extended format, a signature declaration specifying the set of function symbols and their arities is added. In this format, every symbol appearing in the rules must be declared as a function symbol or a variable. Formally, the *trs* declaration in the basic format is replaced by

```
trs     ::= [(VAR idlist)] (SIG funlist) (RULES rulelist)
            [(COMMENT string)]
funlist ::= ϵ | fun funlist
fun     ::= (id int)
```

where *int* is a nonempty sequence of digits. An example of the extended format is provided by COPS #557:

[3] https://www.lri.fr/~marche/tpdb/format.html.

```
(VAR x y z)
(SIG (f 2) (a 0) (b 0) (c 0))
(RULES
  a -> b
  f(x,a) -> f(b,b)
  f(b,x) -> f(b,b)
  f(f(x,y),z) -> f(b,b)
)
(COMMENT
[111] Example 1 with additional constant c
submitted by: Franziska Rapp
)
```

3.2 CTRS Format

The format for first-order conditional rewrite systems is a simplification of the old TPDB format (see Footnote 3), according to the following grammar:

```
ctrs     ::= (CONDITIONTYPE ctype) [(VAR idlist)] (RULES rulelist)
             [(COMMENT string)]
ctype    ::= SEMI-EQUATIONAL | JOIN | ORIENTED
idlist   ::= ϵ | id idlist
rulelist ::= ϵ | rule rulelist
rule     ::= term -> term | term -> term ‘|’ condlist
condlist ::= cond | cond , condlist
cond     ::= term == term
term     ::= id | id() | id(termlist)
termlist ::= term | term , termlist
```

The restrictions on *id* and *string* are the same as in the TRS format. The *ctype* declaration specifies the semantics of the conditions in the rewrite rules: conversion ($\leftrightarrow^*$) for semi-equational CTRSs, joinability ($\downarrow$) for join CTRSs, and reachability ($\rightarrow^*$) for oriented CTRSs. An example of the CTRS format is provided by COPS #488:

```
(CONDITIONTYPE ORIENTED)
(VAR w x y z)
(RULES
  plus(0, y) -> y
  plus(s(x), y) -> s(plus(x, y))
  fib(0) -> pair(0, s(0))
  fib(s(x)) -> pair(z, w) | fib(x) == pair(y, z), plus(y, z) == w
)
(COMMENT
doi:10.4230/LIPIcs.RTA.2015.223
[89] Example 1
submitted by: Thomas Sternagel
)
```

3.3 MSTRS Format

The format for many-sorted term rewrite systems is a modification of the TRS format, according to the following grammar:

```
trs       ::= (SIG funlist) (RULES rulelist) [(COMMENT string)]
funlist   ::= fun | fun funlist
fun       ::= (id sort)
sort      ::= idlist -> id
idlist    ::= ϵ | id idlist
rulelist  ::= ϵ | rule rulelist
rule      ::= term -> term
term      ::= id | id() | id(termlist)
termlist  ::= term | term, termlist
```

The restriction on *id* is the same as in the TRS format. Every term must be a well-typed term according the signature declared in (SIG *funlist*). Symbols (*id*) not declared in *funlist* are variables (which can take any sort). We provide an example (COPS #646):

```
(SIG
   (+      Nat Nat -> Nat)
   (s      Nat -> Nat)
   (0      -> Nat)
   (node   Nat Tree Tree -> Tree)
   (leaf   Nat -> Tree)
   (sum    Tree -> Nat)
)
(RULES
   sum(leaf(x)) -> x
   sum(node(x,yt,zt)) -> +(x,+(sum(yt),sum(zt)))
   +(x,0) -> x
   +(x,s(y)) -> s(+(x,y))
   node(x,yt,zt) -> node(x,zt,yt)
)
(COMMENT
[125] Example 13
submitted by: Takahito Aoto
)
```

3.4 HRS Format

This format deals with higher-order rewrite systems (HRSs) described by Mayr and Nipkow [5] with small modifications detailed below the typing rules. The format follows the same style as the first-order formats, adding type declarations to variables and function symbols as well as syntax for abstraction and application according to the following grammar:

```
hrs        ::= signature (RULES rulelist) [(COMMENT string)]
signature  ::= (VAR sig) (FUN sig) | (FUN sig) (VAR sig)
sig        ::= ϵ | id : type sig
type       ::= type -> type | id | (type)
rulelist   ::= ϵ | rule | rule, rulelist
rule       ::= term -> term
term       ::= id | term(termlist) | term term | \idlist.term | (term)
termlist   ::= term | term, termlist
idlist     ::= id | id idlist
```

In `(FUN` *sig*`)` the function symbols of the HRS are declared, while `(VAR` *sig*`)` declares the types of the variables that are used in the rules. An identifier must not occur in both the `(FUN` *sig*`)` and `(VAR` *sig*`)` sections, but all identifiers that occur in the `(RULES` *rulelist*`)` section must occur in one of them. To save parentheses the following standard conventions are used: In *type*, `->` associates to the right. For terms, application associates to the left, while abstraction associates to the right. Moreover abstractions extend as far to the right as possible, i.e., application binds stronger than abstraction. The algebraic notation *term*(*termlist*) is syntactic sugar for nested application, i.e., `t(u,...,v)` is syntactic sugar for `(... (t u) ...) v`; note that due to left-associativity of application, `s t(u,v)` = `(s t)(u,v)` = `(((s t) u) v)`. Finally, the expression `\x ... y.s` abbreviates `\x. ... \y.s`. Terms must be typable according to the following rules:

$$\frac{\mathtt{x} : \sigma \in \mathtt{VAR}}{\mathtt{x} : \sigma} \qquad \frac{\mathtt{f} : \sigma \in \mathtt{FUN}}{\mathtt{f} : \sigma} \qquad \frac{\mathtt{t} : \sigma \to \tau \qquad \mathtt{u} : \sigma}{\mathtt{t\ u} : \tau} \qquad \frac{\mathtt{x} : \sigma \in \mathtt{VAR} \qquad \mathtt{t} : \tau}{\mathtt{\backslash x.t} : \sigma \to \tau}$$

Terms are modulo $\alpha\beta\eta$. In the interest of user-friendliness and readability we demand that the rules are given in β-normal form, but do not impose any restrictions concerning η. Note that the list of variables declared in `(VAR` *sig*`)` is not exhaustive, fresh variables of arbitrary type are available to construct terms. Left- and right-hand sides of a rewrite rule must be of the same base type, but we do not demand that free variables appearing on the right also occur on the left. An example of the HRS format is provided by COPS #747:

```
(FUN
  app : arrab -> a -> b
  lam : (Va -> b) -> arrab
  var : Va -> a
)
(VAR
  x : Va
  M : a -> b
  N : a
  L : arrab
)
(RULES
  app(lam(\x.M (var x)), N) -> M N,
  lam(\x.app(L, (var x))) -> L
)
(COMMENT
simply-typed lambda calculus with beta/eta in the style of [137,138]
submitted by: Makoto Hamana
)
```

4 Competition

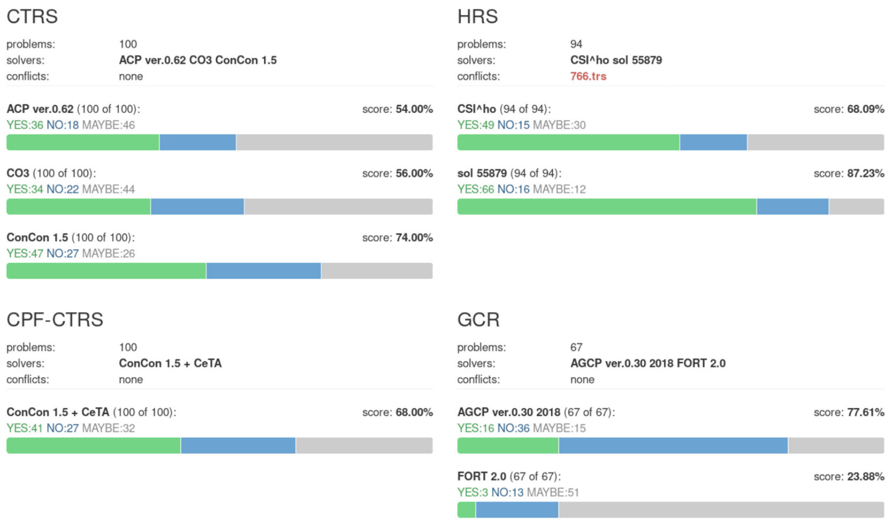
Since 2012 a total of 17 tools participated in CoCo. Many of the tools participated in multiple categories. The proceedings of the International Workshop on Confluence[4] contain (short) descriptions of the contenders. For each category, 100 problems are randomly selected from COPS. Problem selection for CoCo 2019 is subject to the following constraints. For the TRS, CPF-TRS, NFP, UNC, and UNR categories, problems in TRS format are selected. The problems for the SRS category are further restricted to those having the `srs` tag. For the CTRS and CPF-CTRS categories, problems must be in CTRS format and have the tags `3_ctrs` and `oriented`, since participating tools handle only *oriented* CTRSs of *type 3*. In an oriented CTRS the conditions in the rules are interpreted as reachability and type 3 is a syntactic restriction on the distribution of variables in rewrite rules which ensure that rewriting does not introduce fresh variables [6]. For the GCR category, eligible problems must be in TRS or MSTRS format. Being in HRS format is a prerequisite for problems to be selected for the HRS category. For the new COM and INF categories, problems must have the `commutation` and `infeasibility` tags, respectively. The respective formats are described in the next section. New in 2019 is the possibility for tool authors to submit *secret* problems just before the competition. These will be included in the selected problems.

Earlier editions of CoCo only considered problems stemming from the *literature*. This restriction was put in place to avoid bias towards one particular tool

[4] http://cl-informatik.uibk.ac.at/iwc/index.php.

or technique. Since both COPS and CoCo have grown and diversified considerably since their inception, this restriction has become hard to maintain in a meaningful way, while at the same time losing its importance. Consequently it has been dropped for CoCo 2019. Further selection details are available from the CoCo website.

Since 2013 CoCo is executed on the cross-community competition platform StarExec [1]. Each tool has access to a single node and is given 60 s per problem. For a given problem, tools must answer YES or NO, followed by a justification that is understandable by a human expert; any other output signals that the tool could not determine the status of the problem. The possibility in StarExec to reserve a large number of computing nodes allows to complete CoCo within a single slot of a workshop or conference. This live event of CoCo is shared with the audience via the *LiveView* [4] tool which continuously polls new results from StarExec while the competition is running. A screenshot of the LiveView of CoCo 2018 is shown in Fig. 2. New is the realtime display of YES/NO conflicts. Since all categories deal with undecidable problems, and developing software tools is error-prone, conflicts appear once a while. In the past they were identified after the live competition finished, now action by the SC can be taken before winners are announced. As can be seen from the screenshot, in last year's competition there was a YES/NO conflict in the HRS category, which led to lively discussion about the semantics of the HRS format. After each competition, the results are made available from the results page.[5]



CTRS

problems: 100
solvers: **ACP ver.0.62 CO3 ConCon 1.5**
conflicts: none

ACP ver.0.62 (100 of 100): score: **54.00%**
YES:36 NO:18 MAYBE:46

CO3 (100 of 100): score: **56.00%**
YES:34 NO:22 MAYBE:44

ConCon 1.5 (100 of 100): score: **74.00%**
YES:47 NO:27 MAYBE:26

HRS

problems: 94
solvers: **CSI^ho sol 55879**
conflicts: **766.trs**

CSI^ho (94 of 94): score: **68.09%**
YES:49 NO:15 MAYBE:30

sol 55879 (94 of 94): score: **87.23%**
YES:66 NO:16 MAYBE:12

CPF-CTRS

problems: 100
solvers: **ConCon 1.5 + CeTA**
conflicts: none

ConCon 1.5 + CeTA (100 of 100): score: **68.00%**
YES:41 NO:27 MAYBE:32

GCR

problems: 67
solvers: **AGCP ver.0.30 2018 FORT 2.0**
conflicts: none

AGCP ver.0.30 2018 (67 of 67): score: **77.61%**
YES:16 NO:36 MAYBE:15

FORT 2.0 (67 of 67): score: **23.88%**
YES:3 NO:13 MAYBE:51

Fig. 2. Part of the LiveView of CoCo 2018 upon completion.

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

The certification categories (CPF-TRS and CPF-CTRS) are there to ensure that tools produce correct answers. In these categories tools have to produce certified (non-)confluence proofs with their answers. The predominant approach to achieve this uses a combination of confluence prover and independent certifier. First the confluence prover analyses confluence as usual, restricting itself to criteria supported by the certifier. If it is successful the prover prints its proof in the certification problem format (CPF),[6] which is then checked by the certifier. To ensure correctness of this check, soundness of the certifier is mechanized in a proof assistant like Isabelle/HOL. So far only one certifier has participated in CoCo: CeTA.[7]

5 New Categories in 2019

5.1 Commutation

TRSs $\mathcal{R}$ and $\mathcal{S}$ *commute* if the inclusion ${}_{\mathcal{R}}^{*}\!\leftarrow \cdot \rightarrow_{\mathcal{S}}^{*} \subseteq \rightarrow_{\mathcal{S}}^{*} \cdot {}_{\mathcal{R}}^{*}\!\leftarrow$ holds. Commutation is an important generalization of confluence: Apart from direct applications in rewriting, e.g. for confluence,[8] standardization, normalization, and relative termination, commutation is the basis of many results in computer science, like correctness of program transformations [7], and bisimulation up-to [8].

Currently, commutation is supported by the tools CoLL [9] and FORT [10]. The former supports commutation versions of three established confluence techniques: development closedness [11], rule labeling [12], and an adaption of a confluence modulo result by Jouannaud and Kirchner [13]. The latter is a decision tool for the first-order theory of rewriting based on tree automata techniques, but restricted to left-linear right-ground TRSs.

Commutation problems consist of two TRSs $\mathcal{R}$ and $\mathcal{S}$. The question to be answered is whether these commute. To ensure compatibility of the signatures of the involved TRSs, we rename function symbols and variables in $\mathcal{S}$ on demand. Before we describe this precisely, we give an example of a commutation problem that illustrates the problem.

Consider COPS #82 (consisting of the rewrite rules $\mathsf{f}(\mathsf{a}) \rightarrow \mathsf{f}(\mathsf{f}(\mathsf{a}))$ and $\mathsf{f}(x) \rightarrow \mathsf{f}(\mathsf{a})$) and COPS #80 (consisting of $\mathsf{a} \rightarrow \mathsf{f}(\mathsf{a},\mathsf{b})$ and $\mathsf{f}(\mathsf{a},\mathsf{b}) \rightarrow \mathsf{f}(\mathsf{b},\mathsf{a})$). Since function symbol f is unary in COPS #82 and binary in COPS #80, it is renamed to f' in COPS #80:

[6] http://cl-informatik.uibk.ac.at/software/cpf/.
[7] http://cl-informatik.uibk.ac.at/software/ceta/.
[8] The union of confluent, pairwise commuting rewrite systems is confluent.

```
(PROBLEM COMMUTATION)
(COMMENT COPS 82 80)
(VAR x)
(RULES
  f(a) -> f(f(a))
  f(x) -> f(a)
)
(VAR )
(RULES
  a -> f'(a,b)
  f'(a,b) -> f'(b,a)
)
```

The correct answer of this commutation problem is `YES` since the critical peak of $\mathcal{R}$ and $\mathcal{S}$ makes a decreasing diagram [12]. In COPS this problem is given as

```
(PROBLEM COMMUTATION)
(COPS 82 80)
(COMMENT this comment will be removed)
```

and an inlining tool generates the earlier problem before it is passed to tools participating in the commutation category. In general, commutation problems are incorporated into COPS as follows:

```
(PROBLEM COMMUTATION)
(COPS number1 number2)
(COMMENT string)
```

where *number1* and *number2* refer to existing problems in TRS format. The `(COMMENT` *string*`)` declaration is optional. To ensure that their union is a proper TRS, the inlining tool renames function symbols in COPS #*number2* that appear as variable or as function symbol with a different arity in COPS #*number1* by adding a prime (′). The same holds for variables in COPS #*number2* that occur as function symbol in COPS #*number1*.

5.2 Infeasibility Problems

Infeasibility problems originate from different sources. Critical pairs in a conditional rewrite system are equipped with conditions. If no satisfying substitution for the variables in the conditions exists, the critical pair is harmless and can be ignored when analyzing confluence of the rewrite system in question. In this case the critical pair is said to be *infeasible* [14, Definition 7.1.8]. Sufficient conditions for infeasibility of conditional critical pairs are reported in [15,16].

Another source of infeasibility problems is the dependency graph in termination analysis of rewrite systems [17]. An edge from dependency pair $\ell_1 \to r_1$ to dependency pair $\ell_2 \to r_2$ exists in the dependency graph if two substitutions σ and τ can be found such that $r_1\sigma$ rewrites to $\ell_2\tau$. (By renaming the variables in the dependency pairs apart, a single substitution suffices.) If no substitutions exists, there is no edge, which may ease the task of proving termination of the underlying rewrite system [18,19].

We give two examples. The first one stems from the conditional critical pair between the two conditional rewrite rules in COPS #547:

```
(PROBLEM INFEASIBILITY)
(COMMENT COPS 547)
(CONDITIONTYPE ORIENTED)
(VAR x)
(RULES
  f(x) -> a | x == a
  f(x) -> b | x == b
)
(VAR x)
(CONDITION x == a, x == b)
```

The correct answer of this infeasibility problem is `YES` since no term in the underlying conditional rewrite system rewrites to both `a` and `b`. In COPS this problem is given as

```
(PROBLEM INFEASIBILITY)
(COPS 547)
(VAR x)
(CONDITION x == a, x == b)
(COMMENT
doi:10.4230/LIPIcs.FSCD.2016.29
[90] Example 3
submitted by: Raul Gutierrez and Salvador Lucas
)
```

and an inlining tool generates the earlier problem before it is passed to tools participating in the infeasibility category.

The second example is a special case since the condition in the infeasibility problem contains no variables:

```
(PROBLEM INFEASIBILITY)
(COMMENT COPS 47)
(VAR x)
(RULES
  F(x,x) -> A
  G(x) -> F(x,G(x))
  C -> G(C)
)
(CONDITION G(A) == A)
```

has `YES` as correct answer since the term `G(A)` does not rewrite to `A`. This answer can be used to conclude that the underlying rewrite system is not confluent. Again, in COPS this problem is rendered as

```
(PROBLEM INFEASIBILITY)
(COPS 47)
(CONDITION G(A) == A)
(COMMENT this comment will be removed)
```

In general, infeasibility problems are incorporated into COPS as follows:

```
(PROBLEM INFEASIBILITY)
(COPS number)
(VAR idlist)
(CONDITION condlist)
(COMMENT string)
```

where

```
condlist ::= cond | cond , condlist
    cond ::= term == term
```

has the same syntax as the conditional part of a conditional rewrite rule and *number* refers to an existing problem in CTRS or TRS format. If it is a CTRS then the semantics of `==` is the same as declared in the `(CONDITIONTYPE ctype)` declaration of the CTRS; if it is a TRS then the semantics of `==` is `ORIENTED` (reachability, $\to^*$). Variables declared in *idlist* are used as variables in *condlist*. The `(VAR idlist)` declaration can be omitted if the terms in *condlist* are ground. Common function symbols occurring in COPS #*number* and *condlist* have the same arity. Moreover, function symbols in COPS #*number* do not occur as variables in `(VAR idlist)` and function symbols in *condlist* do not occur as variables in COPS #*number*.

5.3 String Rewriting

String rewrite systems (SRSs) are special TRSs in which terms are strings. To ensure that the infrastructure developed for TRSs can be reused, we use the TRS format with the restriction that all function symbols are unary. So a string rule $\mathsf{ab} \to \mathsf{ba}$ is rendered as $\mathsf{a}(\mathsf{b}(x)) \to \mathsf{b}(\mathsf{a}(x))$ where x is a variable. A concrete example (COPS #442) is given below:

```
(VAR x)
(RULES
  f(f(x)) -> x
  f(x) -> f(f(x))
)
(COMMENT
doi:10.4230/LIPIcs.RTA.2015.257
[81] Example 1
)
```

The correct answer of this problem is `YES` since the addition of the redundant rules [20] `f(x) -> f(f(f(x)))` and `f(x) -> x` makes the critical pairs of the SRS development closed [11].

The SRS category has been established to stimulate further research on confluence of string rewriting. In the Termination Competition[9] there is an active community developing powerful techniques for (relative) termination of SRSs. We anticipate that these are beneficial when applied to confluence analysis.

6 Outlook

In the near future we plan to merge CoCo with COPS and CoCoWeb,[10] a convenient web interface to execute the tools that participate in CoCo without local installation, to achieve a single entry point for confluence problems, tools, and competitions. Moreover, the submission interface of COPS will be extended with functionality to support submitters of new problems as well as the CoCo SC. We anticipate that in the years ahead new categories will be added to CoCo. Natural candidates are rewriting modulo AC and nominal rewriting.

Acknowledgments. We are grateful to Nao Hirokawa for continuous support for the infrastructure of CoCo. Fabian Mitterwallner contributed to the inlining and renaming tools for the new commutation and infeasibility categories. Raúl Gutiérrez, Naoki Nishida, and Salvador Lucas contributed the initial set of infeasibility problems (COPS #818 – #936). Johannes Waldmann contributed challenging SRS problems (COPS #987 – #1036). Finally, we acknowledge the TOOLympics 2019 initiators for giving us the opportunity to present CoCo 2019.

References

1. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: a cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS, vol. 8562, pp. 367–373. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_28

[9] http://termination-portal.org/wiki/Termination_Competition.
[10] http://cocoweb.uibk.ac.at/.

2. Aoto, T., Hirokawa, N., Nagele, J., Nishida, N., Zankl, H.: Confluence competition 2015. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS, vol. 9195, pp. 101–104. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_5
3. Aoto, T., et al.: Confluence competition 2018. In: Proceedings of 3rd International Conference on Formal Structures for Computation and Deduction. LIPIcs, vol. 108, pp. 32:1–32:5 (2018). https://doi.org/10.4230/LIPIcs.FSCD.2018.32
4. Hirokawa, N., Nagele, J., Middeldorp, A.: Cops and CoCoWeb: infrastructure for confluence tools. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS, vol. 10900, pp. 346–353. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94205-6_23
5. Mayr, R., Nipkow, T.: Higher-order rewrite systems and their confluence. Theor. Comput. Sci. **192**(1), 3–29 (1998). https://doi.org/10.1016/S0304-3975(97)00143-6
6. Middeldorp, A., Hamoen, E.: Completeness results for basic narrowing. Appl. Algebr. Eng. Commun. Comput. **5**, 213–253 (1994). https://doi.org/10.1007/BF01190830
7. Huet, G.: Confluent reductions: abstract properties and applications to term rewriting systems. J. ACM **27**(4), 797–821 (1980). https://doi.org/10.1145/322217.322230
8. Pous, D.: New up-to techniques for weak bisimulation. Theor. Comput. Sci. **380**(1), 164–180 (2007). https://doi.org/10.1016/j.tcs.2007.02.060
9. Shintani, K., Hirokawa, N.: CoLL: a confluence tool for left-linear term rewrite systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS, vol. 9195, pp. 127–136. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_8
10. Rapp, F., Middeldorp, A.: FORT 2.0. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS, vol. 10900, pp. 81–88. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94205-6_6
11. van Oostrom, V.: Developing developments. Theor. Comput. Sci. **175**(1), 159–181 (1997). https://doi.org/10.1016/S0304-3975(96)00173-9
12. Aoto, T.: Automated confluence proof by decreasing diagrams based on rule-labelling. In: Proceedings of 21st RTA. LIPIcs, vol. 6, pp. 7–16 (2010). https://doi.org/10.4230/LIPIcs.RTA.2010.7
13. Jouannaud, J.P., Kirchner, H.: Completion of a set of rules modulo a set of equations. SIAM J. Comput. **15**(4), 1155–1194 (1986). https://doi.org/10.1137/0215084
14. Ohlebusch, E.: Advanced Topics in Term Rewriting. Springer, New York (2002). https://doi.org/10.1007/978-1-4757-3661-8
15. Lucas, S., Gutiérrez, R.: Use of logical models for proving infeasibility in term rewriting. Inf. Process. Lett. **136**, 90–95 (2018). https://doi.org/10.1016/j.ipl.2018.04.002
16. Sternagel, T., Middeldorp, A.: Infeasible conditional critical pairs. In: Proceedings of 4th International Workshop on Confluence, pp. 13–17 (2015)
17. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theor. Comput. Sci. **236**, 133–178 (2000). https://doi.org/10.1016/S0304-3975(99)00207-8
18. Giesl, J., Thiemann, R., Schneider-Kamp, P.: Proving and disproving termination of higher-order functions. In: Gramlich, B. (ed.) FroCoS 2005. LNCS, vol. 3717, pp. 216–231. Springer, Heidelberg (2005). https://doi.org/10.1007/11559306_12

19. Middeldorp, A.: Approximating dependency graphs using tree automata techniques. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, pp. 593–610. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45744-5_49
20. Nagele, J., Felgenhauer, B., Middeldorp, A.: Improving automatic confluence analysis of rewrite systems by redundant rules. In: Proceedings of 26th RTA. LIPIcs, vol. 36, pp. 257–268 (2015)