infChecker
A Tool for Checking Infeasibility*

Raúl Gutiérrez† Salvador Lucas
Valencian Research Institute for Artificial Intelligence
Universitat Politècnica de València
Camino de Vera s/n, E-46022 Valencia, Spain

{rgutierrez,slucas}@dsic.upv.es

Given a Conditional Term Rewriting System (CTRS) $\mathscr{R}$ and terms s and t, we say that the reachability condition $s \to^* t$ is *feasible* if there is a substitution σ instantiating the variables in s and t such that the *reachability test* $\sigma(s) \to^*_{\mathscr{R}} \sigma(t)$ succeeds; otherwise, we call it *infeasible*. Checking infeasibility of such (sequences of) reachability conditions is important in the analysis of computational properties of CTRSs, like confluence or operational termination. Recently, a logic-based approach to prove and disprove infeasibility has been introduced. In this paper we introduce a new framework for checking feasibility/infeasibility and a new tool, infChecker, which is based on such an approach.

1 Introduction

When analyzing the computational behavior of CTRSs $\mathscr{R}$, consisting of rules $\ell \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$, we need to consider two kinds of computations:

1. the reduction of expressions in the usual way, i.e., by replacing an instance $\sigma(\ell)$ of the left-hand side ℓ by the instance $\sigma(r)$ of the right-hand side r using a matching substitution σ and
2. the evaluation of the conditions $s_i \to^* t_i$ in the rules, which (for *oriented* CTRSs) are treated as reachability tests $\sigma(s_i) \to^* \sigma(t_i)$.

In this setting, representing rewriting steps in CTRSs as proofs of goals in the logic of (oriented) CTRSs with inference system in Figure 1 becomes a natural way to represent computations [7]. Given a CTRS $\mathscr{R}$, an inference system $\mathscr{I}(\mathscr{R})$ is obtained from the inference rules in Figure 1 by *specializing* $(C)_{f,i}$ for each k-ary symbol f in the signature $\mathscr{F}$ and $1 \le i \le k$ and $(Rl)_\rho$ for all conditional rules $\rho : \ell \to r \Leftarrow c$ in $\mathscr{R}$.
We write $s \to_{\mathscr{R}} t$ (resp. $s \to^*_{\mathscr{R}} t$) iff there is a proof tree for $s \to t$ (resp. $s \to^* t$) using the inference system $\mathscr{I}(\mathscr{R})$, whose rules are *schematic* in the sense that each inference rule $\frac{B_1 \;\cdots\; B_n}{A}$ can be used under any *instance* $\frac{\sigma(B_1) \;\cdots\; \sigma(B_n)}{\sigma(A)}$ of the rule by a substitution σ. Therefore, the fact that a term s rewrites to a term t is witnessed by the existence of a *proof* in the inference system $\mathscr{I}(\mathscr{R})$ of the goal $s \to t$ (one step) or $s \to^* t$ (zero or more steps).

Example 1. *Consider the following CTRS $\mathscr{R}$ (903.trs[1]) that computes the minimum of a list of natural numbers using Peano notation:*

*Partially supported by the EU (FEDER), and projects RTI2018-094403-B-C32, PROMETEO/2019/098, and SP20180225.
†Raúl Gutiérrez was also supported by INCIBE program "Ayudas para la excelencia de los equipos de investigación avanzada en ciberseguridad".

[1]This problem belongs to the database COPS of confluence problems in http://cops.uibk.ac.at/

Submitted to:
PROLE 2019

$$\text{(R)}\quad \frac{}{x \to^* x} \qquad \text{(C)}_{f,i}\quad \frac{x_i \to y_i}{f(x_1,\ldots,x_i,\ldots,x_k) \to f(x_1,\ldots,y_i,\ldots,x_k)} \text{ for all } f \in \mathscr{F}^{(k)} \text{ and } 1 \le i \le k$$

$$\text{(T)}\quad \frac{x \to y \qquad y \to^* z}{x \to^* z} \qquad \text{(Rl)}_{\rho}\quad \frac{s_1 \to^* t_1 \quad \cdots \quad s_n \to^* t_n}{\ell \to r} \text{ for } \rho : \ell \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n \in \mathscr{R}$$

Figure 1: Inference rules for conditional rewriting with a CTRS $\mathscr{R}$ with signature $\mathscr{F}$

$$\begin{array}{rcl} le(0,s(y)) & \to & true \\ le(s(x),s(y)) & \to & le(x,y) \\ le(x,0) & \to & false \\ min(cons(x,nil)) & \to & x \\ min(cons(x,xs)) & \to & x \Leftarrow min(xs) \to^* y, le(x,y) \to^* true \\ min(cons(x,xs)) & \to & y \Leftarrow min(xs) \to^* y, le(x,y) \to^* false \end{array}$$

We can write $min(cons(0,cons(s(0),nil))) \to_{\mathscr{R}} 0$ because we can generates the following (partially depicted) proof tree:

$$\dfrac{\dfrac{\cdots}{min(cons(s(0),nil)) \to^* s(0)} \qquad \dfrac{\cdots}{le(0,s(0)) \to^* true}}{min(cons(0,cons(s(0),nil))) \to 0}$$

In the logic of CTRSs, a first-order theory $\overline{\mathscr{R}}$ associated to $\mathscr{R}$, where $\to$ and $\to^*$ are seen as predicates, can be obtained from $\mathscr{I}(\mathscr{R})$: the inference rules $\frac{B_1 \cdots B_n}{A}$ in $\mathscr{I}(\mathscr{R})$ are considered as *sentences* $(\forall x_1,\ldots,x_m)\, B_1 \wedge \cdots \wedge B_n \Rightarrow A$, where $\{x_1,\ldots,x_m\}$ is the (possibly empty) set of variables occurring in the atoms $B_1,\ldots,B_n$ and A. If such a set is empty, we write $B_1 \wedge \cdots \wedge B_n \Rightarrow A$.

Example 2. *The first-order theory $\overline{\mathscr{R}}$ for $\mathscr{R}$ in Example 1 is:*

$$(\forall x)\ x \to^* x \tag{1}$$
$$(\forall x,y,z)\ x \to y \wedge y \to^* z \Rightarrow x \to^* z \tag{2}$$
$$(\forall x,y)\ x \to y \Rightarrow s(x) \to s(y) \tag{3}$$
$$(\forall x,y,z)\ x \to y \Rightarrow cons(x,z) \to cons(y,z) \tag{4}$$
$$(\forall x,y,z)\ x \to y \Rightarrow cons(z,x) \to cons(z,y) \tag{5}$$
$$(\forall x,y,z)\ x \to y \Rightarrow le(x,z) \to le(y,z) \tag{6}$$
$$(\forall x,y,z)\ x \to y \Rightarrow le(z,x) \to le(z,y) \tag{7}$$
$$(\forall x,y)\ x \to y \Rightarrow min(x) \to min(y) \tag{8}$$
$$(\forall y)\ le(0,s(y)) \to true \tag{9}$$
$$(\forall x,y)\ le(s(x),s(y)) \to le(x,y) \tag{10}$$
$$(\forall x)\ le(x,0) \to false \tag{11}$$
$$(\forall x)\ min(cons(x,nil)) \to x \tag{12}$$
$$(\forall x,y,xs)\ min(xs) \to^* y \wedge le(x,y) \to^* true \Rightarrow min(cons(x,xs)) \to x \tag{13}$$
$$(\forall x,xs)\ min(xs) \to^* y \wedge le(x,y) \to^* false \Rightarrow min(cons(x,xs)) \to y \tag{14}$$

Deductions with $\overline{\mathscr{R}}$ proceed *à la Hilbert* by using *modus ponens* and *generalization* as inference rules, the usual set of *logical axioms*, and $\overline{\mathscr{R}}$ as the set of *proper* axioms [11, Section 2.3]. In this way, we can also prove goals $s \to t$ and $s \to^* t$, written $\overline{\mathscr{R}} \vdash s \to t$ and $\overline{\mathscr{R}} \vdash s \to^* t$, respectively. For all terms s and t, we have (i) $s \to_{\mathscr{R}} t$ iff $\overline{\mathscr{R}} \vdash s \to t$ and (ii) $s \to^*_{\mathscr{R}} t$ iff $\overline{\mathscr{R}} \vdash s \to^* t$.

In this paper, we present the tool infChecker, a tool for proving and disproving feasibility conditions taking advantage of this logical approach. Section 3 presents the notion of feasibility condition. Section 4 describes the feasibility framework and the processors used by our tool. Section 5 provides some details about the implementation, the web interface and the strategy used. Finally, Section 6 shows the experimental evaluation and Section 7 concludes.

2 Preliminaries

We use the standard notations in term rewriting (see, e.g., [12]). In this paper, $\mathscr{X}$ denotes a countable set of *variables* and $\mathscr{F}$ denotes a *signature*, i.e., a set of *function symbols* $\{f,g,\ldots\}$, each with a fixed *arity* given by a mapping $ar : \mathscr{F} \to \mathbb{N}$. The set of terms built from $\mathscr{F}$ and $\mathscr{X}$ is $\mathscr{T}(\mathscr{F},\mathscr{X})$. The symbol labeling the root of t is denoted as $root(t)$. The set of variables occurring in t is $\mathscr{V}ar(t)$. Terms are viewed as labeled trees in the usual way. *Positions* $p,q,\ldots$ are represented by chains of positive natural numbers used to address subterms $t|_p$ of t. The *set of positions* of a term t is $\mathscr{P}os(t)$. A substitution is a mapping from variables into terms which is homomorphically extended to a mapping from terms to terms. A conditional rule is written $\ell \to r \Leftarrow s_1 \to^* t_1, \cdots, s_n \to^* t_n$, where $\ell, r, s_1, t_1, \ldots, s_n, t_n \in \mathscr{T}(\mathscr{F},\mathscr{X})$ and $\ell \notin \mathscr{X}$. As usual, ℓ and r are called the left- and right-hand sides of the rule, and the sequence $s_1 \to^* t_1, \cdots, s_n \to^* t_n$ (often abbreviated to c) is the *conditional part* of the rule. We often write $s_i \to^* t_i \in c$ to refer to the i-th atomic condition in c or $s \to t \in c$ if the position of the atomic condition in c does not matter. Rules $\ell \to r \Leftarrow c$ are classified according to the distribution of variables as follows: type 1 (or 1-rules), if $\mathscr{V}ar(r) \cup \mathscr{V}ar(c) \subseteq \mathscr{V}ar(\ell)$; type 2, if $\mathscr{V}ar(r) \subseteq \mathscr{V}ar(\ell)$; type 3, if $\mathscr{V}ar(r) \subseteq \mathscr{V}ar(\ell) \cup \mathscr{V}ar(c)$; and type 4, if no restriction is given. A CTRS $\mathscr{R}$ is a set of conditional rules; $\mathscr{R}$ is called an n-CTRS if it contains only n-rules; A 3-CTRS $\mathscr{R}$ is called *deterministic* if for each rule $\ell \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$ in $\mathscr{R}$ and each $1 \le i \le n$, we have $\mathscr{V}ar(s_i) \subseteq \mathscr{V}ar(\ell) \cup \bigcup_{j=1}^{i-1} \mathscr{V}ar(t_j)$.

By a *structure* $\mathscr{A}$ for a first-order language we mean an interpretation of the function and predicate symbols ($f,g,\ldots$ and $P,Q,\ldots$, respectively) as mappings $f^{\mathscr{A}}, g^{\mathscr{A}}, \ldots$ and relations $P^{\mathscr{A}}, Q^{\mathscr{A}}, \ldots$ on a given set (carrier) also denoted $\mathscr{A}$. Then, the usual interpretation of first-order formulas with respect to the structure is considered. A model for a set $\mathscr{S}$ of first-order sentences (i.e., formulas whose variables are all *quantified*) is just a structure that makes them all true, written $\mathscr{A} \models \mathscr{S}$. By *correctness* of the first-order predicate calculus, for all models $\mathscr{A}$ of a theory $\mathscr{S}$, whenever $\mathscr{S} \vdash \varphi$ holds for a sentence φ, we have $\mathscr{A} \models \varphi$.

3 Feasibility conditions

Given a CTRS $\mathscr{R}$ and terms s and t, we say that the feasibility condition $s \to^* t$ is *$\mathscr{R}$-feasible* (or just *feasible* if no confusion arises) if there is a substitution σ instantiating the variables in s and t such that the *reachability test* $\sigma(s) \to^*_{\mathscr{R}} \sigma(t)$ succeeds; otherwise, we call it *infeasible*. As in [6, Definition 2], sequences $\mathscr{G} = (s_i \to^* t_i)_{i=1}^n$, where $n > 0$, are called *feasibility sequences*. We say that $\mathscr{G}$ is *$\mathscr{R}$-feasible* if there is a substitution such that all the reachability tests $\sigma(s_i) \to^*_{\mathscr{R}} \sigma(t_i)$ are satisfied. In [5, 6], we presented an approach to deal with infeasibility using a satisfiability criterion: a sequence $\mathscr{G}$ as above is

infeasible if the first-order theory $\overline{\mathscr{R}}$ together with the *negation* of the sentence

$$(\exists \vec{x}) \bigwedge_{i=1}^{n} s_i \to^* t_i \tag{15}$$

where $\vec{x}$ are the variables occurring in terms s_i and t_i for $1 \leq i \leq n$, is *satisfiable* by some interpretation $\mathscr{A}$ of the function and predicate symbols, i.e., $\mathscr{A} \models \overline{\mathscr{R}} \cup \{\neg(15)\}$ holds [6, Theorem 6]. Actually, as showed in [4], if (15) is a logical consequence of $\overline{\mathscr{R}}$ (i.e., $\overline{\mathscr{R}} \vdash (15)$ holds), then the feasibility of $\mathscr{G}$ is *proved*. Thus, this logical approach provides a sound and complete method to (dis)prove feasibility.

Example 3. *Continuing with Example 1, the following sequence:*

$$min(nil) \to^* x, le(y,x) \to^* true$$

corresponds to the following first-order formula (15):

$$(\exists x,y)\, min(nil) \to^* x \wedge le(y,x) \to^* true$$

In the following section, we show how to deal with feasibility sequences in practice.

4 Feasibility Framework

In order to automatically analyze whether a sequence $\mathscr{G}$ is *feasible* or *infeasible*, we describe a framework similar to the one presented in [1] for termination purposes. We define a notion of problem and processor related to our settings and show how to apply these processors in order to get a sound or complete proof.

Definition 4 (fProblem and fProcessor)**.** *An* fProblem τ *is a pair* $\tau = (\mathscr{R}, \mathscr{G})$*, where* $\mathscr{R}$ *is a CTRS and* $\mathscr{G}$ *is a sequence* $(s_i \to^* t_i)_{i=1}^n$*. The* fProblem τ *is* feasible *if* $\mathscr{G}$ *is* $\mathscr{R}$*-feasible; otherwise it is* infeasible*.*

An fProcessor P *is a partial function from fProblems into sets of fProblems. Alternatively, it can return* "yes"*.* $\mathscr{D}om(\mathsf{P})$ *represents the domain of* P*, i.e., the set of fProblems* τ *that* P *is defined for.*

An fProcessor P *is* sound *if for all* $\tau \in \mathscr{D}om(\mathsf{P})$*,* τ *is feasible whenever either* $\mathsf{P}(\tau) =$ "yes" *or* $\exists \tau' \in \mathsf{P}(\tau)$*, such that* τ' *is feasible.*

An fProcessor P *is* complete *if for all* $\tau \in \mathscr{D}om(\mathsf{P})$*,* τ *is infeasible whenever* $\forall \tau' \in \mathsf{P}(\tau)$*,* τ' *is infeasible.*

Feasibility problems can be proved or disproved by using a proof tree as follows.

Definition 5 (Feasibility Proof Tree)**.** *Let* $\tau = (\mathscr{R}, \mathscr{G})$ *be an fProblem. A feasibility proof tree* $\mathscr{T}$ *for* τ *is a tree whose inner nodes are labeled with fProblems and the leaves are labeled with fProblems,* "yes" *or* "no"*. The root of* $\mathscr{T}$ *is labeled with* τ *and for every inner node n labeled with* τ'*, there is a fProcessor* P *such that* $\tau' \in \mathscr{D}om(\mathsf{P})$ *and:*

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

In this way, a feasibility proof tree is obtain by the combination of different fProcessors.

Theorem 6 (Feasibility Framework)**.** *Let* $\mathscr{R}$ *be a CTRS,* $\mathscr{G}$ *be a feasibility sequence, and* $\mathscr{T}$ *be a feasibility proof tree for* $\tau_I = (\mathscr{R}, \mathscr{G})$*. Then:*

1. *if all leaves in $\mathscr{T}$ are labeled with* "no" *and all involved fProcessors are complete for the fProblems they are applied to, then $\mathscr{G}$ is $\mathscr{R}$-infeasible.*
2. *if $\mathscr{T}$ has a leaf labeled with* "yes" *and all fProcessors in the path from τ_I to the leaf are sound for the fProblems they are applied to, then $\mathscr{G}$ is $\mathscr{R}$-feasible.*

Proof. We analyze the two cases:

1. Since all leaves in $\mathscr{T}$ are labeled with "no", by definition of infeasibility proof tree the nodes n that immediately precede the leaves are labeled with fProblems τ that are *infeasible* (because $\mathsf{P}(\tau) = \emptyset$ for some complete fProcessor P). By definition of completeness of the fProcessors involved in the construction of $\mathscr{T}$, the root of $\mathscr{T}$ is labeled with a *infeasible* fProblem. Therefore, τ_I is *infeasible* and, by definition, $\mathscr{G}$ is $\mathscr{R}$-infeasible.
2. If $\mathscr{T}$ contains a leaf L labeled with "yes", then there is a *sound* processor P such that, for the node n (with label τ_f) that immediately precedes L, we have $\mathsf{P}(\tau_f) =$ "yes". Hence, τ_f is *feasible*. Since all processors used on the path from the root to L are also sound, τ is also *feasible* and, by definition, $\mathscr{G}$ is $\mathscr{R}$-feasible.

□

In the following subsections we describe a number of sound and complete fProcessors. If is clear from the context, we use processor and problem instead of fProcessor and fProblem.

4.1 Satisfiability fProcessor

In [6], we described a satisfiability approach to prove infeasibility.

Theorem 7 ([6, Theorem 6])**.** *Let $\mathscr{R}$ be a CTRS, $(s_i \to_i^* t_i)_{i=1}^n$ be a feasibility sequence, and $\mathscr{A}$ be a structure with nonempty domain. If $\mathscr{A} \models \overline{\mathscr{R}} \cup \{\neg(\exists \vec{x}) \bigwedge_{i=1}^n s_i \to_i^* t_i\}$, then the sequence is $\mathscr{R}$-infeasible.*

In this theorem, we prove that if we can find a model that satisfies the first-order theory $\overline{\mathscr{R}}$ together with the negated first-order formula that represents the feasibility sequence $\mathscr{G}$, we can prove the $\mathscr{R}$-infeasibility of the sequence. The processor described in this section integrates this satisfiability approach to prove infeasibility in our framework.

We use the definitions below to extract the set of usable rules for reachability in an fProblem. When the feasibility sequence $\mathscr{G}$ satisfies that all s_i, such that $s_i \to^* t_i \in \mathscr{G}$, are ground terms we can reduce the set of rules from $\mathscr{R}$ that are needed to generate the model. Discarding rules from $\mathscr{R}$ we can obtain simpler models.

$$RULES(\mathscr{R},t) \quad = \quad \{\ell \to r \Leftarrow c \in R \mid \exists p \in \mathscr{P}os(t), root(\ell) = root(t|_p)\}$$

The set of *usable rules* for a term t is:

$$\mathscr{U}(\mathscr{R},t) = RULES(\mathscr{R},t) \cup \bigcup_{l\to r\Leftarrow c \in RULES(\mathscr{R},t)} \left(\mathscr{U}(\mathscr{R}^\sharp, r) \cup \bigcup_{s\to^* t \in c} \mathscr{U}(\mathscr{R}^\sharp, s) \right)$$

where $\mathscr{R}^\sharp = \mathscr{R} - RULES(\mathscr{R},t)$. Given an fProblem $(\mathscr{R},\mathscr{G})$, we let

$$\mathscr{U}(\mathscr{R},\mathscr{G}) = \begin{cases} \bigcup_{s_i \to^* t_i \in \mathscr{G}} \mathscr{U}(\mathscr{R}, s_i) & \text{if all } s_i \text{ in } \mathscr{G} \text{ are ground} \\ R & \text{otherwise} \end{cases}$$

Example 8. *Consider the sequence* $\mathscr{G} : (\exists x)\, le(0,s(0)) \to^* x$*, the set of usable rules* $\mathscr{U}(\mathscr{R},\mathscr{G})$ *only contains the le-rules, discarding the rules that cannot be applied in the sequence.*

By [6, Proposition 4], $\mathscr{R}$-infeasibility of $\mathscr{G}$ can be proved as $(\bigcup_{s_i \to^* t_i \in \mathscr{G}} \mathscr{U}(\mathscr{R}, s_i))$-infeasibility provided that all terms s_i are ground.

Theorem 9 (Satisfiability fProcessor)**.** *Let* $\tau = (\mathscr{R},\mathscr{G})$ *be an fProblem with* $\mathscr{G} = (s_i \to^* t_i)_{i=1}^n$*. Let* $\mathscr{A}$ *be a structure such that* $\mathscr{A} \neq \emptyset$ *and* $\mathscr{A} \models \overline{\mathscr{U}(\mathscr{R},\mathscr{G})} \cup \{\neg(\exists \vec{x}) \bigwedge_{i=1}^n s_i \to^* t_i\}$*. The fProcessor* $\mathsf{P}^{\mathsf{Sat}}$ *given by* $\mathsf{P}^{\mathsf{Sat}}(\tau) = \emptyset$ *is sound and complete.*

Proof. By [6, Proposition 4] and Theorem 7. □

In infChecker, we use the model generators AGES [2] and Mace4 [10] to find suitable structures $\mathscr{A}$ to be used in the implementation of $\mathsf{P}^{\mathsf{Sat}}$.

Example 10. *Given* $\mathscr{R}$ *from Example 1 and the negation of* $\mathscr{G}$ *from Example 3:*

$$(\forall x,y)\, \neg(min(nil) \to^* x) \vee \neg(le(y,x) \to^* true)$$

we can solve $\tau_I = (\mathscr{R},\mathscr{G})$ *by applying* $\mathsf{P}^{\mathsf{Sat}}$ *and obtaining* $\mathsf{P}^{\mathsf{Sat}}(\tau_I) = \emptyset$ *using* AGES *with the following interpretation:*

```
Domain:

|N U {-1}

Function Interpretations:

[le(x,y)] = y
[min(x)] = x
[0] = 1
[cons(x,y)] = 1 + x + y
[false] = 1
[nil] =  - 1
[s(x)] = 1 + x
[true] = 0

Predicate Interpretations:

x ->* y <=> (x >= y)
x -> y <=> ((x >= y) /\ (1 + y >= 0))
```

4.2 Provability fProcessor

The following processor integrates the logic-based approach to program analysis described in [4] to prove feasibility by theorem proving.

Theorem 11 (Provability fProcessor)**.** *Let* $\tau = (\mathscr{R},\mathscr{G})$ *be an fProblem with* $\mathscr{G} = (s_i \to^* t_i)_{i=1}^n$ *such that* $\overline{\mathscr{R}} \vdash (\exists \vec{x}) \bigwedge_{i=1}^n s_i \to^* t_i$ *holds. The fProcessor* $\mathsf{P}^{\mathsf{Prov}}$ *given by* $\mathsf{P}^{\mathsf{Prov}}(\tau) =$ "yes" *is sound and complete.*

Proof. If $\overline{\mathscr{R}} \vdash (\exists \vec{x}) \bigwedge_{i=1}^{n} s_i \to^* t_i$ holds, then $(\exists \vec{x}) \bigwedge_{i=1}^{n} s_i \to^* t_i$ is a logical consequence of $\overline{\mathscr{R}}$. Consider the structure $\mathscr{A}$ with domain $\mathscr{T}(\mathscr{F},\mathscr{X})$, k-ary function symbols $f \in \mathscr{F}$ interpreted by $f^{\mathscr{A}}(t_1,\ldots,t_k) = f(t_1,\ldots,t_k)$, and predicate symbols $\to$ and $\to^*$ interpreted as the one-step and many-step rewriting on $\mathscr{T}(\mathscr{F},\mathscr{X})$, respectively (i.e., $\to_{\mathscr{R}}$ and $\to^*_{\mathscr{R}}$). Clearly, $\mathscr{A}$ is a model of $\overline{\mathscr{R}}$ and therefore it is a model of $(\exists \vec{x}) \bigwedge_{i=1}^{n} s_i \to^* t_i$ as well. This means that there are terms $u_1,\ldots,u_k$ such that $\sigma(s_i) \to^* \sigma(t_i)$ for the substitution σ defined by $\sigma(x_i) = u_i$ for all $1 \le i \le k$. Thus, $\mathscr{G}$ is feasible. □

In infChecker, we use the theorem prover Prover9 [10] as a backend to implement the use of $\mathsf{P}^{\mathsf{Prov}}$.

Example 12. *Given $\mathscr{R}$ from Example 1 and the sequence of feasibility conditions $\mathscr{G}$ (836.trs):*

$$le(x,min(y)) \to^* false, min(y) \to^* x$$

that can be seen as the following first-order formula:

$$(\exists x,y)\ le(x,min(y)) \to^* false \wedge min(y) \to^* x \tag{16}$$

For $\tau_I = (\mathscr{R},\mathscr{G})$, we have $\mathsf{P}^{\mathsf{Prov}}(\tau_I) =$ "yes" by using Prover9 *to prove (16) as follows:*

```
(17) -(le(x,min(y)) ->* false) | -(min(y) ->* x) # [deny(16)]
(18) le(x,0) ->* false [ur(2,11,1)]
(19) -(le(min(x),min(x)) ->* false) [resolve(17,1)]
(20) -(le(min(x),min(x)) -> y) | -(y ->* false) [resolve(19,2)]
(21) -(le(min(x),y) ->* false) | -(min(x) -> y) [resolve(20,7)]
(22) -(le(min(cons(x,nil)),x) ->* false) [resolve(21,12)]
(23) $F [resolve(22,18)]
```

4.3 Narrowing on Feasibility Conditions fProcessor

Reachability problems $\sigma(s) \to^* \sigma(t)$ are often investigated using *narrowing* and unification conditions directly over terms s and t, thus avoiding the 'generation' of the required substitution σ. In this section, we use narrowing to simplify feasibility conditions in $\mathscr{G}$. Definition 13 describes how narrowing is defined in the context of CTRSs. In the following, we write $s =^?_\theta t$ if s and t unifies with *mgu* θ.

Definition 13. [9, Definition 79] *Let $\mathscr{R}$ be a CTRS. A term s* narrows *to a term t (written $s \rightsquigarrow_{\mathscr{R},\theta,p} t$ or just $s \rightsquigarrow_{\mathscr{R},\theta} t$ or even $s \rightsquigarrow t$), iff there is a nonvariable position $p \in \mathscr{P}os_{\mathscr{F}}(s)$, a* renamed *rule $\ell \to r \Leftarrow s_1 \to t_1,\ldots,s_n \to t_n$ in $\mathscr{R}$, substitutions $\theta_0,\ldots,\theta_n,\tau_1,\ldots,\tau_n$, and terms $t'_1,\ldots,t'_n$ such that:*

1. *$s|_p =^?_{\theta_0} \ell$,*
2. *for all i, $1 \le i \le n$, $\eta_{i-1}(s_i) \rightsquigarrow^*_{\mathscr{R},\theta_i} t'_i$ and $t'_i =^?_{\tau_i} \theta_i(\eta_{i-1}(t_i))$, where $\eta_0 = \theta_0$ and for all $i > 0$, $\eta_i = \tau_i \circ \theta_i \circ \eta_{i-1}$, and*
3. *$t = \theta(s[r]_p)$, where $\theta = \eta_n$.*

We write $u \rightsquigarrow^*_{\mathscr{R},\beta} v$ for terms u,v and substitution β iff there are terms $u_1,\ldots,u_{m+1}$ and substitutions $\beta_1,\ldots,\beta_m$ for some $m \ge 0$ such that

$$u = u_1 \rightsquigarrow_{\mathscr{R},\beta_1} u_2 \rightsquigarrow_{\mathscr{R},\beta_2} \cdots \rightsquigarrow_{\mathscr{R},\beta_m} u_{m+1} = v$$

and $\beta = \beta_{m-1} \circ \cdots \circ \beta_1$ (or $\beta = \varepsilon$ if $m = 0$).

Given a term u, the set $N_1(\mathscr{R},u)$ represents the set of one-step $\mathscr{R}$-narrowings issued from u:

$$N_1(\mathscr{R},u) = \{(v,\theta{\downarrow_{\mathscr{V}ar(u)}}) \mid u \rightsquigarrow_{\ell\to r\Leftarrow c,\theta} v, \ell \to r \Leftarrow c \in \mathscr{R}\} \tag{17}$$

where $\theta{\downarrow_{\mathscr{V}ar(u)}}$ is a substitution defined by $\theta{\downarrow_{\mathscr{V}ar(u)}}(x) = \theta(x)$ if $x \in \mathscr{V}ar(u)$ and $\theta{\downarrow_{\mathscr{V}ar(u)}}(x) = x$ otherwise.

As discussed in [9, Section 7.5], the set $N_1(\mathscr{R},u)$ is not computable in many cases or even *infinite*. In [9, Proposition 87] some sufficient conditions for finiteness of $N_1(\mathscr{R},u)$ are given. Knowing these restrictions, in Theorem 15 we define a narrowing processor on feasibility conditions.

Given a feasibility sequence $\mathscr{G} = (s_i \to^* t_i)_{i=1}^n$, $\overline{\mathscr{N}}(\mathscr{R},\mathscr{G},i)$ returns a new set of feasibility sequences where each element of the set corresponds to a possible narrowing on the condition i.

$$\overline{\mathscr{N}}(\mathscr{R},\mathscr{G},i) = \{\mathscr{G}[\vec{\theta}, w \to^* t_i]_i \mid s_i \to^* t_i \in \mathscr{G}, (w,\theta) \in N_1(\mathscr{R},s_i)\}$$

where $\vec{\theta}$ consists of new conditions $x_1 \to^* \theta(x_1), \ldots, x_m \to^* \theta(x_m)$ obtained from the bindings in θ for variables in $\mathscr{V}ar(s_i) = \{x_1,\ldots,x_m\}$.

Definition 14 (Narrowing on Feasibility Conditions fProcessor)**.** *Let $\tau = (\mathscr{R},\mathscr{G})$ be an fProblem, $s_i \to^* t_i \in \mathscr{G}$, and $\mathscr{N} \subseteq \overline{\mathscr{N}}(\mathscr{R},\mathscr{G},i)$ finite.* P^{NC} *is given by* $\mathsf{P}^{\mathsf{NC}}(\tau) = \{(\mathscr{R},\mathscr{N})\}$.

Given a term s, we let $NRules(\mathscr{R},s)$ the set of rules $\alpha : \ell \to r \Leftarrow c \in \mathscr{R}$ such that a nonvariable subterm t of s is a *narrex* of α, and $\theta{\downarrow_{\mathscr{V}ar(s)}}$ is a substitution defined by $\theta{\downarrow_{\mathscr{V}ar(s)}}(x) = \theta(x)$ if $x \in \mathscr{V}ar(s)$ and $\theta{\downarrow_{\mathscr{V}ar(s)}}(x) = x$ otherwise.

Theorem 15. P^{NC} *is sound. If $\mathscr{N} = \overline{\mathscr{N}}(\mathscr{R},\mathscr{G},i)$ and $s_i \to^* t_i \in \mathscr{G}$ is such that s_i and t_i do* not *unify and either s_i is* ground *or (1) $NRules(\mathscr{R},s_i)$ is a TRS and (2) s_i is linear, then* P^{NC} *is complete.*

Proof. Similar to [8, Theorem 20].

□

Note that even when $\overline{\mathscr{N}}(\mathscr{R},\mathscr{G},i)$ is infinite, a subset $\mathscr{N}$ of $\overline{\mathscr{N}}(\mathscr{R},\mathscr{G},i)$ can be sufficient to prove feasibility. However, to prove infeasibility we need to consider all the possible narrowings.

Example 16. *Consider the following CTRS $\mathscr{R}$ (896.trs):*

$$\begin{array}{rcll}
add(0,x) & \to & x & \\
add(s(x),y) & \to & s(add(x,y)) & \\
div(0,s(x)) & \to & 0 & \\
div(s(x),s(y)) & \to & 0 & \Leftarrow\ lte(s(x),y) \to^* true \\
div(s(x),s(y)) & \to & s(q) & \Leftarrow\ lte(s(x),y) \to^* false, \\
 & & & \quad\ div(minus(x,y),s(y)) \to^* q \\
 & & & \\
lte(0,y) & \to & true & \\
lte(s(x),0) & \to & false & \\
lte(s(x),s(y)) & \to & lte(x,y) & \\
minus(0,s(y)) & \to & 0 & \\
minus(s(x),s(y)) & \to & minus(x,y) & \\
minus(x,0) & \to & x & \\
mod(0,y) & \to & 0 & \\
mod(x,0) & \to & x & \\
mod(x,s(y)) & \to & mod(minus(x,s(y)),s(y)) & \Leftarrow\ lte(s(y),x) \to^* true \\
mod(x,s(y)) & \to & x & \Leftarrow\ lte(s(y),x) \to^* false \\
mult(0,y) & \to & 0 & \\
mult(s(x),y) & \to & add(mult(x,y),y) &
\end{array}$$

$$\begin{array}{lcll} power(x,0) & \to & s(0) & \\ power(x,n) & \to & mult(mult(y,y),s(0)) & \Leftarrow \; n \to^* s(n'), mod(n,s(s(0))) \to^* 0, \\ & & & \quad power(x,div(n,s(s(0)))) \to^* y \\ power(x,n) & \to & mult(mult(y,y),x) & \Leftarrow \; n \to^* s(n'), mod(n,s(s(0))) \to^* s(z), \\ & & & \quad power(x,div(n,s(s(0)))) \to^* y \end{array}$$

together with $\mathscr{G}$*:*

$$\{lte(s(x),0) \to^* true\}$$

Let $\tau_I = (\mathscr{R},\mathscr{G})$ *be our initial fProblem. We can safely apply* P^{NC} *to the condition* $lte(s(x),0) \to^* true$ *because* $NRules(\mathscr{R},lte(s(x),0))$ *is a TRS (it contains only the* lte *rules) and* $lte(s(x),0)$ *is linear. Therefore,*

$$\mathsf{P}^{\mathsf{NC}}(\tau_I) = \{\tau_1\}$$

where $\overline{\mathscr{N}}(\mathscr{R},\mathscr{G},1) = \{\{x \to^* x, false \to^* true\}\}$ *and* $\tau_1 = (\mathscr{R},\{x \to^* x, false \to^* true\})$*. Now, our new set of sequences is easier to prove than our original sequence and by applying* $\mathsf{P}^{\mathsf{Sat}}$ *to the resulting problem, we obtain* $\mathsf{P}^{\mathsf{Sat}}(\tau_1) = \emptyset$ *by using* Mace4 *with the following interpretation:*

```
Domain:

{0,1}

Function Interpretations:

[false] = 0
[true] = 1
[0] = 0
[s](0) = 0          [s](1) = 1
[add](0,0) = 0      [add](0,1) = 1      [add](1,0) = 0      [add](1,1) = 1
[div](0,0) = 0      [div](0,1) = 0      [div](1,0) = 0      [div](1,1) = 0
[lte](0,0) = 1      [lte](0,1) = 1      [lte](1,0) = 1      [lte](1,1) = 1
[minus](0,0) = 0    [minus](0,1) = 0    [minus](1,0) = 1    [minus](1,1) = 1
[mod](0,0) = 0      [mod](0,1) = 0      [mod](1,0) = 1      [mod](1,1) = 1
[mult](0,0) = 0     [mult](0,1) = 1     [mult](1,0) = 0     [mult](1,1) = 1
[power](0,0) = 0    [power](0,1) = 0    [power](1,0) = 1    [power](1,1) = 1

Predicate Interpretations:

0 ->* 0 = true      0 ->* 1 = false     1 ->* 0 = true      1 ->* 1 = true
0 -> 0 = true       0 -> 1 = false      1 -> 0 = true       1 -> 1 = true
```

In this case, our model contains two values, $true$ *is interpreted with the value* 1 *and* $false$ *with the value* 0*. Furthermore,* $0 \to^* 1$ *is not satisfied by the model. The resulting proof tree is shown in Figure 2.*

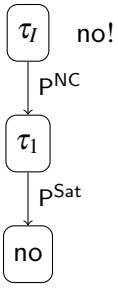
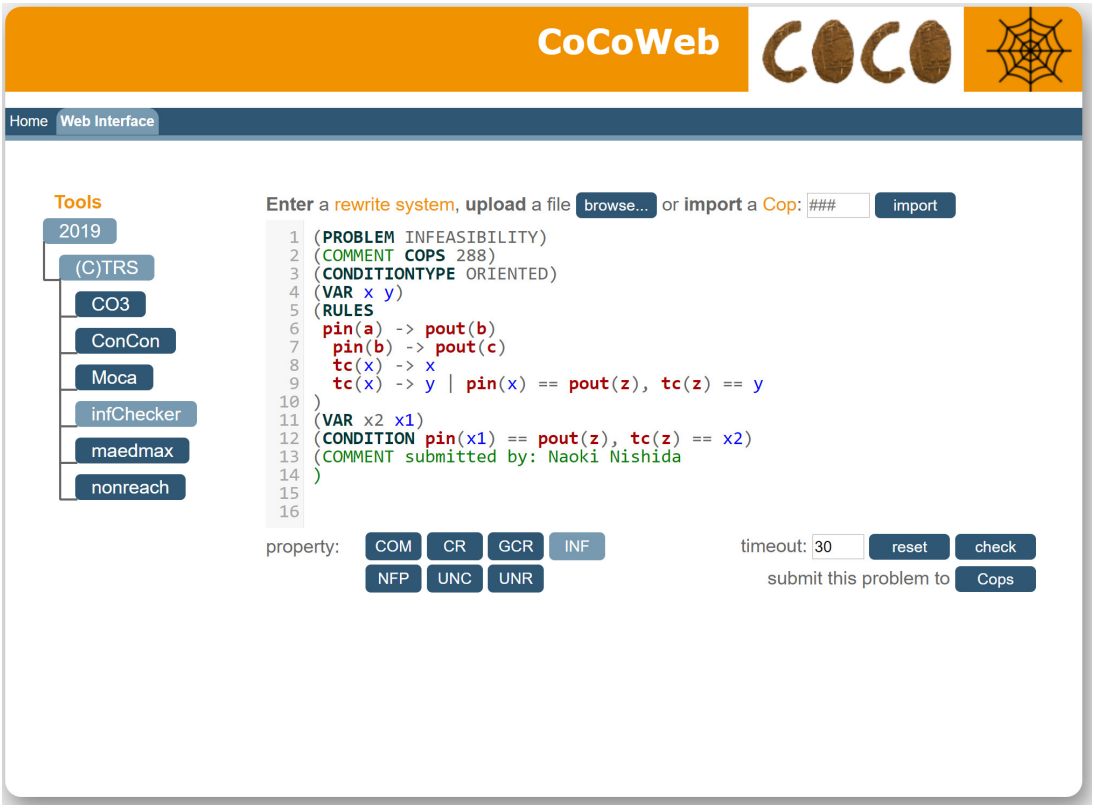


Figure 2: Proof tree obtained from Example 16

5 Implementation and Web Interface

infChecker 1.0 is written in Haskell[2] and consists of 28 new Haskell modules with more than 3500 lines of code. The tool is accessible from the web interface of the CoCoWeb[3] platform [3] together with other infeasibility tools:



The input format is an extended version of the well-known TPDB format, which is the official format of in the *infeasibility* (INF) category of the Confluence Competition (CoCo)[4]. This extended version allows to declare a list of feasibility conditions in the block identified with the reserved word `CONDITION`.

[2] See `http://haskell.org/`.
[3] Available at http://cl-informatik.uibk.ac.at/software/cocoweb/
[4] See `http://project-coco.uibk.ac.at/2019/categories/infeasibility.php`

Currently, infChecker only implements the three processors presented in Section 4. When the "check" button is pressed, the proof strategy used by infChecker is the following:

1. we try to prove feasibility using $\mathsf{P}^{\mathsf{Prov}}$;
2. if $\mathsf{P}^{\mathsf{Prov}}$ fails, we apply $\mathsf{P}^{\mathsf{Sat}}$;
3. if $\mathsf{P}^{\mathsf{Sat}}$ fails, we apply P^{NC};
4. if P^{NC} succeeds and modifies the feasibility sequence, we go to Item 2, otherwise we return MAYBE.

If the answer is YES or NO, the output of the tool is a proof in plain text of the (in)feasibility of the input feasibility sequence.

6 Experimental Evaluation

We participated in the INF category of the 2019 Confluence Competition (CoCo)[5]. Participants have a limit of 60 seconds for each input program to return a proof of feasibility or infeasibility (or a *don't know* answer). infChecker obtained the following results:

INF Tool	Yes	No	Total
infChecker	40	32	72
nonreach	30	0	30
Moca	26	0	26
maedmax	15	0	15
CO3	12	0	12

Note that answers *Yes/No* in the table refer to *infeasibility* problems (which is the focus of the competition). In our setting, given a CTRS $\mathscr{R}$ and an infeasibility problem given as a feasibility sequence $\mathscr{G}$, we just return *Yes* if τ_I is proved infeasible, and *No* if τ_I is proved feasible.

Apart from the 32 negative answers, there are 7 more examples that can be proved positively using infChecker only. Furthermore, there are 10 examples that can be proved by other tools and cannot be proved by infChecker. Thanks to representing CTRS specifications together with the sequence of feasibility conditions as first-order theories we are not only the most successful tool for checking infeasibility, but the only tool which is currently able to disprove infeasibility (by proving feasibility).

7 Conclusions and Future Work

In this paper we present infChecker, a new tool for checking feasibility conditions of CTRSs that takes advantage of the logic-based approach presented in [4, 5, 6]. We successfully participated in the 2019 Confluence Competition in the INF (infeasibility) category, being the most powerful tool for checking both infeasibility and feasibility.

Currently, the tool has only three processors. As a subject for future work, we would like to increase the power of the tool with new processors focused on feasibility conditions, analyze the examples that cannot be proved by infChecker but can be proved by other tools and extend the tool to deal with more involved rewrite systems (order-sorted, axioms...).

[5] http://project-coco.uibk.ac.at/2019/

References

[1] J. Giesl, R. Thiemann, P. Schneider-Kamp & S. Falke (2006): *Mechanizing and Improving Dependency Pairs*. *Journal of Automatic Reasoning* 37(3), pp. 155–203.

[2] R. Gutiérrez & S. Lucas (2019): *Automatic Generation of Logical Models with AGES (System Description)*. In P. Fontaine, editor: *Proc. of the 27th International Conference on Automated Deduction, CADE 2019*, *LNCS* , Springer, p. to appear.

[3] N. Hirokawa, J. Nagele & A. Middeldorp (2018): *Cops and CoCoWeb: Infrastructure for Confluence Tools*. In D. Galmiche, S. Schulz & R Sebastiani, editors: *Proc. of the 9th International Joint Conference on Automated Reasoning, IJCAR'18*, Springer, pp. 346–353.

[4] S. Lucas (2019): *Proving Program Properties as First-Order Satisfiability*. In F. Mesnard & P. J. Stuckey, editors: *LOPSTR 2018: Logic-Based Program Synthesis and Transformation*, *LNCS* 11408, Springer, pp. 3–21.

[5] S. Lucas & R. Gutiérrez (2017): *A Semantic Criterion for Proving Infeasibility in Conditional Rewriting*. In B. Accattoli & B. Felgenhauer, editors: *Proc. of the 6th International Workshop on Confluence, IWC'17*, pp. 15–19.

[6] S. Lucas & R. Gutiérrez (2018): *Use of Logical Models for Proving Infeasibility in Term Rewriting*. *Information Processing Letters* 136, pp. 90–95.

[7] S. Lucas, C. Marché & J. Meseguer (2005): *Operational termination of conditional term rewriting systems*. *Information Processing Letters* 95(4), pp. 446–453.

[8] S. Lucas, J. Meseguer & R. Gutiérrez: *The 2D Dependency Pair Framework for conditional rewrite systems. Part II: Advanced processors and implementation techniques.* *Journal of Automated Reasoning* in revision.

[9] S. Lucas, J. Meseguer & R. Gutiérrez (2018): *The 2D Dependency Pair Framework for conditional rewrite systems. Part I: Definition and basic processors.* *Journal of Computer and System Sciences* 96, pp. 74–106.

[10] W. McCune: *Prover9 and Mace4*. [online]. Available at `https://www.cs.unm.edu/~mccune/mace4/`.

[11] E. Mendelson (1997): *Introduction to Mathematical Logic*, fourth edition. Chapman & Hall.

[12] E. Ohlebusch (2002): *Advanced Topics in Term Rewriting*. Springer.