FORT 2.0[⋆]

Franziska Rapp[⋆⋆] and Aart Middeldorp

Department of Computer Science, University of Innsbruck, Austria
{franziska.rapp,aart.middeldorp}@uibk.ac.at

Abstract. FORT is a tool that implements the first-order theory of rewriting for the decidable class of left-linear right-ground rewrite systems. It can be used to decide properties of a given rewrite system and to synthesize rewrite systems that satisfy arbitrary properties expressible in the first-order theory of rewriting. In this paper we report on the extensions that were incorporated in the latest release (2.0) of FORT. These include witness generation for existentially quantified variables in formulas, support for combinations of rewrite systems, as well as an extension to deal with non-ground terms for properties related to confluence.

1 Introduction

In a recent paper [6] we introduced FORT, a decision and synthesis tool for the first-order theory of rewriting induced by finite left-linear right-ground rewrite systems. In this theory one can express well-known properties like termination, normalization, and confluence, but also properties like strong confluence ($\forall s\, \forall t\, \forall u\, (s \to t \wedge s \to u \implies \exists v\, (t \to^{=} v \wedge u \to^{*} v))$) and the normal form property ($\forall s\, \forall t\, \forall u\, (s \to t \wedge s \to^{!} u \implies t \to^{!} u)$). The decision procedure implemented in FORTis based on tree automata techniques (Dauchet and Tison [3]).

In this paper we present several extensions designed to make the tool more useful. First of all, we added support for combinations of rewrite systems. This is required to express properties like *commutation* ($\forall s \forall t \forall u\, (s \to_0^* t \wedge s \to_1^* u \implies \exists v\, (t \to_1^* v \wedge u \to_0^* v))$) and *equivalence* ($\forall s\, \forall t\, (s \leftrightarrow_0^* t \iff s \leftrightarrow_1^* t)$) that refer to two or more rewrite systems. Tree automata operate on *ground* terms. Consequently, variables in formulas range over ground terms and hence the properties that FORT is able to decide are restricted to ground terms. Whereas for termination and normalization this makes no difference, for other properties it does, even for the restricted class of *left-linear right-ground rewrite systems* as will be shown below. This brings us to the second extension: How can one use FORT to decide properties on open terms? We show that for properties related to confluence it suffices to add one or two fresh constants. We furthermore provide sufficient conditions which obviate the need for additional constants. The third extension is concerned with increasing the understanding of the yes/no answer

[⋆] This research is supported by FWF (Austrian Science Fund) project P30301.
[⋆⋆] Currently at Allgemeines Rechenzentrum Innsbruck.

provided by FORT in decision mode. For logical formulas with free variables we are not only interested whether they are satisfied by a particular rewrite system, but also which terms act as witnesses. Witness generation is also of interest for existentially quantified variables appearing in formulas.

We assume familiarity with term rewriting. A command-line version of FORT is available.[1] We refer to [6] and the description on the website for the syntax of the commands and formulas that can be passed to FORT.

2 Witness Generation

The usual output of FORT consists of tree automata (or their size) corresponding to subformulas of the given formula, which is hard to read. To help the user in understanding why a property holds or does not hold, we have now implemented witness generation, which provides evidence by generating an n-tuple of ground terms for free variables or (implicitly) existentially quantified ones. For instance, if a given or synthesized TRS is not ground-confluent ($\neg \forall s \forall t \forall u : (s \to^* t \wedge s \to^* u \implies \exists v \, (t \to^* v \wedge u \to^* v)$), it is interesting to provide witnessing terms for the variables s, t, and u. Given the TRS consisting of the rules $\mathsf{a} \to \mathsf{f}(\mathsf{a}, \mathsf{b})$ and $\mathsf{f}(\mathsf{a}, \mathsf{b}) \to \mathsf{f}(\mathsf{b}, \mathsf{a})$, FORT produces the following terms as witnesses: $s = \mathsf{f}(\mathsf{a}, \mathsf{b})$, $t = \mathsf{f}(\mathsf{b}, \mathsf{a})$, and $u = \mathsf{f}(\mathsf{f}(\mathsf{a}, \mathsf{b}), \mathsf{b})$.

To cope with n-ary relations on terms, FORT uses bottom-up tree automata that operate on encodings of n-tuples of ground terms, subsequently called RR_n automata. Given a signature $\mathcal{F}$ we let $\mathcal{F}^{(n)} = (\mathcal{F} \cup \{\bot\})^n$ with $\bot \notin \mathcal{F}$ a fresh constant. The arity of a symbol $f_1 \cdots f_n \in \mathcal{F}^{(n)}$ is the maximum of the arities of $f_1, \ldots, f_n$. Given terms $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$, the encoding $\langle t_1, \ldots, t_n \rangle \in \mathcal{T}(\mathcal{F}^{(n)})$ is best illustrated on a concrete example. For the ground terms $s = \mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{f}(\mathsf{b}, \mathsf{b}))$, $t = \mathsf{g}(\mathsf{g}(\mathsf{a}))$, and $u = \mathsf{f}(\mathsf{b}, \mathsf{g}(\mathsf{a}))$ we have $\langle s, t, u \rangle = \mathsf{fgf}(\mathsf{ggb}(\mathsf{aa}\bot), \mathsf{f}\bot\mathsf{g}(\mathsf{b}\bot\mathsf{a}, \mathsf{b}\bot\bot))$. So for each position occurring in one of the terms, function symbols of all terms are put together, where the fresh symbol $\bot$ is used for missing positions. We refer to [6] for a formal definition.

The recursive algorithm depicted in Figure 1 generates (encoded) witnesses that reach a given state α of an RR_n automaton. As a side condition, it does not make use of the given set Q_v of visited states to avoid non-termination. In the outer call, $Q_v = \varnothing$. The set $\mathcal{C}$ of candidates contains all transition rules ending in the given state α such that the states in the left-hand side of the rule were not visited before. Furthermore, in the outermost call ($Q_v = \{\alpha\}$) rules having a $\bot$ in their list of function symbols are excluded as well, since they do not produce encodings of terms over the original signature $\mathcal{F}$. Then a rule with minimal number of arguments (to obtain small witnesses) is chosen from $\mathcal{C}$ and the function `find_terms` is called recursively for each argument position to get witnesses for the argument states. This might fail in case the automaton was not normalized beforehand and we end up in non-reachable states, in which case we move on to the next candidate rule from $\mathcal{C}$.

[1] http://cl-informatik.uibk.ac.at/software/FORT

Input: • RR$_n$ automaton $\mathcal{A} = (\mathcal{F}^{(n)}, Q, Q_f, \Delta)$
• state $\alpha \in Q$, set of states $Q_v \subseteq Q$
Output: • term accepted in α not using states in Q_v

```
$Q_v := Q_v \cup \{\alpha\}$;
$\mathcal{C} := \{\, fs(\alpha_1, \ldots, \alpha_m) \to \alpha \in \Delta \mid \alpha_1, \ldots, \alpha_m \notin Q_v \wedge (Q_v = \{\alpha\} \implies \bot \notin fs)\,\}$;
while $\mathcal{C} \neq \varnothing$ do
    select $fs(\alpha_1, \ldots, \alpha_m) \to \alpha \in \mathcal{C}$ with minimal $m$ and remove it from $\mathcal{C}$;
    if ($t_i$ := find_terms($\mathcal{A}, \alpha_i, Q_v$) for $i = 1, \ldots, m$) does not fail then
        return $fs(t_1, \ldots, t_m)$
od;
fail
```

Fig. 1. Function `find_terms` for witness generation.

In order to apply this algorithm to generate an n-tuple of terms accepted by an RR$_n$ automaton, one has to call the function `find_terms` with a final state of the automaton and decode the resulting term over $\mathcal{F}^{(n)}$.

Example 1. Consider the signature $\mathcal{F} = \{\mathsf{a}\colon 0, \mathsf{b}\colon 0, \mathsf{g}\colon 1, \mathsf{f}\colon 2, \mathsf{h}\colon 3\}$ and the RR$_2$ automaton $\mathcal{A}$ over $\mathcal{F}^{(2)}$ with final state ✓ and transition rules

$$\begin{array}{lllll} \mathsf{aa} \to \alpha & \bot\mathsf{a} \to \alpha' & \mathsf{fg}(\alpha, \beta') \to \gamma & \mathsf{fh}(\alpha, \beta, \alpha') \to \checkmark & \mathsf{ff}(\alpha, \gamma) \to \checkmark \\ \mathsf{bb} \to \beta & \mathsf{b}\bot \to \beta' & \mathsf{gf}(\beta, \alpha') \to \gamma & \mathsf{gg}(\gamma) \to \gamma & \mathsf{ff}(\checkmark, \alpha) \to \checkmark \end{array}$$

We compute `find_terms`$(\mathcal{A}, \checkmark, \varnothing)$. We have $\mathcal{C} = \{\mathsf{fh}(\alpha, \beta, \alpha') \to \checkmark, \mathsf{ff}(\alpha, \gamma) \to \checkmark\}$. Note that $\mathsf{ff}(\checkmark, \alpha) \to \checkmark$ does not belong to $\mathcal{C}$. We select the rule $\mathsf{ff}(\alpha, \gamma) \to \checkmark$ having the least number of arguments. The recursive call `find_terms`$(\mathcal{A}, \alpha, \{\checkmark\})$ returns aa, since $\mathsf{aa} \to \alpha$ is the only rule ending in α. Depending on the selected transition rule ending in γ, after further recursive calls we obtain $\mathsf{fg}(\mathsf{aa}, \mathsf{b}\bot)$ or $\mathsf{gf}(\mathsf{bb}, \bot\mathsf{a})$. The latter term gives rise to $\mathsf{ff}(\mathsf{aa}, \mathsf{gf}(\mathsf{bb}, \bot\mathsf{a}))$, which encodes the pair of witnessing terms $\mathsf{f}(\mathsf{a}, \mathsf{g}(\mathsf{b}))$ and $\mathsf{f}(\mathsf{a}, \mathsf{f}(\mathsf{b}, \mathsf{a}))$.

3 Combinations

Several important properties, like (normalization) equivalence, commutation, and relative termination, refer to two or more TRSs. Inspired by Zantema's work on Carpa [10], we added support for combinations of rewrite systems in FORT 2.0. For instance, the commutation property can be written as

```
forall s, t, u ([0] s ->* t & [1] s ->* u =>
      exists v ([1] t ->* v & [0] u ->* v))
```

in FORT syntax. Here the indices 0 and 1 refer to different TRSs (provided by the user in decision mode). Lists of indices (e.g. `[0,2,3]`) are also supported, indicating that the subsequent (sub)formula is checked for the union of the TRSs corresponding to the listed indices. If no index is specified, the union of all involved TRSs is taken. We return to commutation in Section 6. Here we compare FORT with Carpa. The following task is mentioned in the Carpa distribution.[2]

Example 2. If we want to generate two terminating abstract rewrite systems (ARSs) such that their union is non-terminating, the formula (`[0] SN & [1] SN & ~SN`) can be used. The additional requirement that the composition of both relations is a subset of the transitive closure of one of them is expressed as

```
forall s, t, u ([0] s -> t & [1] t -> u => [0] s ->+ u | [1] s ->+ u)
```

FORT synthesizes the following two ARSs satisfying the conjunction of these requirements: $\mathcal{A}_0 = \{\mathsf{a} \to \mathsf{b}, \mathsf{c} \to \mathsf{a}\}$ and $\mathcal{A}_1 = \{\mathsf{a} \to \mathsf{b}, \mathsf{b} \to \mathsf{c}\}$. Using completely different techniques, the same ARSs are generated by Carpa.

Whereas Carpa is restricted to ARSs, its successor Carpa+ can synthesize TRSs that admit rules of the shape $a \to b$, $a \to f(b)$, and $f(a) \to b$ with exactly one unary function symbol f. The properties supported by Carpa+ are restricted to those that can be encoded into the conjunctive fragment of SMT-LRA (linear real arithmetic). For this reason properties like (local) confluence are only approximated. In Carpa these and many others properties were encoded exactly in SAT, which is possible since the number of different terms (constants) is finite in the case of ARSs.

Small ARSs as in Example 2 are easily synthesized by FORT. Checking the examples from the Carpa website for correctness poses no problem for the decision mode of FORT, but the method does not scale very well in synthesis mode.

4 Properties on Open Terms

Since the decision algorithm implemented in FORT is based on tree automata, variables in formulas range over ground terms and hence the properties that FORT is able to decide are restricted to ground terms. Whereas for properties like termination and normalization (restricted to right-ground rewrite systems) this makes no difference, for most properties it does, even for left-linear right-ground rewrite systems, as illustrated by the following example.

Example 3. The TRS $\mathcal{R}$ consisting of the rewrite rules $\mathsf{a} \to \mathsf{b}$, $\mathsf{f}(x, \mathsf{a}) \to \mathsf{b}$, and $\mathsf{f}(\mathsf{b}, \mathsf{b}) \to \mathsf{b}$ is ground confluent since all ground terms rewrite to the normal form b. However, $\mathcal{R}$ is not confluent as $\mathsf{b} \leftarrow \mathsf{f}(x, \mathsf{a}) \to \mathsf{f}(x, \mathsf{b})$ with normal forms b and $\mathsf{f}(x, \mathsf{b})$.

In this section we consider the following properties of single TRSs:

$$\begin{aligned} \mathsf{CR}: &\quad \forall\, s\, \forall\, t\, \forall\, u\, (s \to^* t \wedge s \to^* u \implies t \downarrow u) \\ \mathsf{WCR}: &\quad \forall\, s\, \forall\, t\, \forall\, u\, (s \to t \wedge s \to u \implies t \downarrow u) \end{aligned}$$

[2] https://www.win.tue.nl/~hzantema/carpa.html

$$\begin{aligned}
\mathsf{SCR}\colon &\quad \forall s\, \forall t\, \forall u\, (s \to t \wedge s \to u \implies \exists v\, (t \to^{=} v \wedge u \to^{*} v)) \\
\mathsf{UN}\colon &\quad \forall s\, \forall t\, \forall u\, (s \to^{!} t \wedge s \to^{!} u \implies t = u) \\
\mathsf{UNC}\colon &\quad \forall t\, \forall u\, (t \leftrightarrow^{*} u \wedge \mathsf{NF}(t) \wedge \mathsf{NF}(u) \implies t = u) \\
\mathsf{NFP}\colon &\quad \forall s\, \forall t\, \forall u\, (s \to t \wedge s \to^{!} u \implies t \to^{!} u)
\end{aligned}$$

The results stated for confluence below apply to commutation as well. Let $\mathfrak{P} = \{\mathsf{CR}, \mathsf{WCR}, \mathsf{SCR}, \mathsf{UN}, \mathsf{UNC}, \mathsf{NFP}\}$. For $P \in \mathfrak{P}$ we write $\mathsf{G}P$ to denote the property P restricted to ground terms. Let $\mathfrak{R}$ consist of all pairs $(\mathcal{F}, \mathcal{R})$ where $\mathcal{R}$ is a finite left-linear right-ground TRSs over the finite signature $\mathcal{F}$ (containing at least one constant).

For all properties $P \in \mathfrak{P}$, $\mathsf{G}P$ does not imply P. Example 3 gives a counterexample to the implication for all properties except SCR. For SCR the TRS consisting of the rules $\mathsf{a} \to \mathsf{b}$, $\mathsf{b} \to \mathsf{f}(\mathsf{a}, \mathsf{a})$, and $\mathsf{f}(\mathsf{a}, x) \to \mathsf{a}$ can be used. The peak $\mathsf{f}(\mathsf{b}, x) \leftarrow \mathsf{f}(\mathsf{a}, x) \to \mathsf{a}$ cannot be joined using $\to^{=} \cdot {}^{*}\!\leftarrow$ but any ground instance of $\mathsf{f}(\mathsf{b}, x)$ can be reached from a. Nevertheless, according to the following result (whose proof can be found in [7]), it is possible to check a property $P \in \mathfrak{P}$ using tree automata techniques.

Lemma 4. *If* $(\mathcal{F}, \mathcal{R}) \in \mathfrak{R}$ *then*

1. $(\mathcal{F}, \mathcal{R}) \vDash P \iff (\mathcal{F} \cup \{c\}, \mathcal{R}) \vDash \mathsf{G}P$ *for all* $P \in \mathfrak{P} \setminus \{\mathsf{UNC}\}$
2. $(\mathcal{F}, \mathcal{R}) \vDash \mathsf{UNC} \iff (\mathcal{F} \cup \{c, c'\}, \mathcal{R}) \vDash \mathsf{GUNC}$

with fresh constants c *and* c'. □

The following example shows that adding a single fresh constant is not sufficient for UNC.

Example 5. The left-linear right-ground TRS $\mathcal{R}$ consisting of the rules

$$\mathsf{a} \to \mathsf{b} \qquad \mathsf{f}(x, \mathsf{a}) \to \mathsf{f}(\mathsf{b}, \mathsf{b}) \qquad \mathsf{f}(\mathsf{b}, x) \to \mathsf{f}(\mathsf{b}, \mathsf{b}) \qquad \mathsf{f}(\mathsf{f}(x, y), z) \to \mathsf{f}(\mathsf{b}, \mathsf{b})$$

does not satisfy UNC since $\mathsf{f}(x, \mathsf{b}) \leftarrow \mathsf{f}(x, \mathsf{a}) \to \mathsf{f}(\mathsf{b}, \mathsf{b}) \leftarrow \mathsf{f}(y, \mathsf{a}) \to \mathsf{f}(y, \mathsf{b})$ is a conversion between distinct normal forms. Adding a single fresh constant c is not enough to violate GUNC as $\mathsf{f}(\mathsf{c}, \mathsf{b})$ is the only ground instance of $\mathsf{f}(x, \mathsf{b})$ that is a normal form. The latter is ensured by the last two rewrite rules. Adding another fresh constant c' solves the issue. FORT 2.0 generates the witnessing terms $\mathsf{f}(\$, \mathsf{b})$ and $\mathsf{f}(\%, \mathsf{b})$: $\mathsf{f}(\$, \mathsf{b}) \leftarrow \mathsf{f}(\$, \mathsf{a}) \to \mathsf{f}(\mathsf{b}, \mathsf{b}) \leftarrow \mathsf{f}(\%, \mathsf{a}) \to \mathsf{f}(\%, \mathsf{b})$. Here $\$$ and $\%$ are the fresh constants added by FORT.

Lemma 4 does not generalize to arbitrary properties that are expressible in the first-order theory of rewriting. Consider for example the formula φ:

$$\neg\, \exists s\, \exists t\, \exists u\, \forall v\, (v \leftrightarrow^{*} s \vee v \leftrightarrow^{*} t \vee v \leftrightarrow^{*} u)$$

which is satisfied on open terms (with respect to any $(\mathcal{F}, \mathcal{R}) \in \mathfrak{R}$). For the TRS consisting of the rule $\mathsf{f}(x) \to \mathsf{a}$ and two additional constants c and c', φ does not hold for ground terms because every ground term is convertible to a, c or c'.

The following result (whose proof can be found in [7]) shows that for properties in $\mathfrak{P}$ it is not always necessary to add fresh constants. Here a *monadic* signature consists of constants and unary function symbols.

Lemma 6. *Let $(\mathcal{F},\mathcal{R}) \in \mathfrak{R}$ such that $\mathcal{R}$ is ground or $\mathcal{F}$ is monadic. For all $P \in \mathfrak{P}$, $(\mathcal{F},\mathcal{R}) \vDash P$ if and only if $(\mathcal{F},\mathcal{R}) \vDash \mathsf{G}P$.* $\square$

FORT indeed benefits from this optimization. For instance, deciding GCR of Cops #506 whose signature is monadic takes 1.73 seconds if a fresh constant is added, compared to 0.85 seconds if Lemma 6 is used.

We now report on some synthesis experiments that we performed in FORT, based on the following diagram which summarizes the relationships between properties P and $\mathsf{G}P$ for $P \in \mathfrak{P}$:

$$\begin{tikzcd}
\mathsf{WCR} \arrow[ddd, Rightarrow] & & \mathsf{CR} \arrow[ll, Rightarrow] \arrow[r, Rightarrow] \arrow[ddd, Rightarrow] & \mathsf{NFP} \arrow[r, Rightarrow] \arrow[ddd, Rightarrow] & \mathsf{UNC} \arrow[r, Rightarrow] \arrow[ddd, Rightarrow] & \mathsf{UN} \arrow[ddd, Rightarrow] \\
& \mathsf{SCR} \arrow[ur, Rightarrow] \arrow[d, Rightarrow] & & & & \\
& \mathsf{GSCR} \arrow[dr, Rightarrow] & & & & \\
\mathsf{GWCR} & & \mathsf{GCR} \arrow[ll, Rightarrow] \arrow[r, Rightarrow] & \mathsf{GNFP} \arrow[r, Rightarrow] & \mathsf{GUNC} \arrow[r, Rightarrow] & \mathsf{GUN}
\end{tikzcd}$$

The following TRSs were produced by FORT on the given formulas when restricting the signature (using the command-line option `-f "f:2 a:0 b:0"`) to a binary function symbol f and two constants a and b:

`GWCR & ~WCR & ~GCR`	$\mathsf{a} \to \mathsf{b}$	$\mathsf{f}(x,\mathsf{a}) \to \mathsf{a}$	$\mathsf{a} \to \mathsf{f}(\mathsf{a},\mathsf{a})$
`GCR & ~CR & ~GSCR`	$\mathsf{a} \to \mathsf{b}$	$\mathsf{f}(x,\mathsf{a}) \to \mathsf{b}$	$\mathsf{b} \to \mathsf{f}(\mathsf{a},\mathsf{a})$
`GNFP & ~NFP & ~GCR`	$\mathsf{a} \to \mathsf{b}$	$\mathsf{f}(x,\mathsf{a}) \to \mathsf{f}(\mathsf{a},\mathsf{a})$	$\mathsf{f}(\mathsf{b},\mathsf{b}) \to \mathsf{f}(\mathsf{a},\mathsf{a})$
`GUNC & ~UNC & ~GNFP`	$\mathsf{a} \to \mathsf{a}$	$\mathsf{f}(x,\mathsf{a}) \to \mathsf{a}$	$\mathsf{f}(\mathsf{b},x) \to \mathsf{b}$

We do not know whether there exist TRSs over the restricted signature that satisfy `GUN & ~UN & ~GUNC`. Human expertise was used to produce a witness over a larger signature, which was subsequently simplified using the decision mode of FORT:

$\mathsf{b} \to \mathsf{a}$	$\mathsf{c} \to \mathsf{c}$	$\mathsf{d} \to \mathsf{c}$	$\mathsf{f}(x,\mathsf{a}) \to \mathsf{A}$	$\mathsf{f}(x,\mathsf{A}) \to \mathsf{A}$
$\mathsf{b} \to \mathsf{c}$		$\mathsf{d} \to \mathsf{e}$	$\mathsf{f}(x,\mathsf{e}) \to \mathsf{A}$	$\mathsf{f}(\mathsf{c},x) \to \mathsf{A}$

FORT produces the following terms as witnesses for the fact that UN is not satisfied: $s = \mathsf{f}(\mathsf{d},\$)$, $t = \mathsf{A}$, and $u = \mathsf{f}(\mathsf{e},\$)$. Indeed both A and $\mathsf{f}(\mathsf{e},\$)$ are normal forms reachable from $\mathsf{f}(\mathsf{d},\$)$. Moreover, we obtain witnesses $t = \mathsf{a}$ and $u = \mathsf{e}$ showing that GUNC does not hold. (The rule $\mathsf{c} \to \mathsf{c}$ is needed to satisfy GUN.)

Since the previous release (1.0) of FORT, many-sorted TRSs are supported. As the set of many-sorted ground terms is accepted by a tree automaton, this extension was mostly straightforward. However, concerning confluence-related properties on non-ground terms, one has to add one (or two for UNC) fresh constant(s) for every sort that variables appearing in the rules can take.

5 Other Extensions

Apart from the extensions detailed before, the efficiency of FORT was improved (in FORT 1.0) using the multithreading features of Java for parallelizing both synthesis and decision. Furthermore, we now also admit variables in right-hand sides of rewrite rules, provided they appear only once in the rule. This extension opens up the possibility of using FORT to compute dependency graphs based on the non-variable approximation for termination analysis [5], check infeasibility of conditional critical pairs for confluence analysis of conditional TRSs [9], and compute needed redexes based on the strong and non-variable approximations for the analysis of optimal normalizing strategies [4]. Even inside FORT this extension was already useful. We previously used $(\nrightarrow\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow_{\mathcal{R}} \cup \nrightarrow\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow_{\mathcal{R}}^{-1})^+$ to construct an RR_2 automaton for the conversion relation ($\leftrightarrow^*$) but now we can use $\nrightarrow\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow_{\mathcal{R} \cup \mathcal{R}^{-1}}^+$ which results in smaller automata for many TRSs. For instance, the conversion relation $\leftrightarrow^*$ induced by the TRS

$$\mathsf{g}(\mathsf{f}(\mathsf{a})) \to \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{a}))) \qquad \mathsf{g}(\mathsf{f}(\mathsf{a})) \to \mathsf{f}(\mathsf{f}(\mathsf{a})) \qquad \mathsf{f}(\mathsf{f}(\mathsf{a})) \to \mathsf{f}(\mathsf{a})$$

is modeled as an RR_2 automaton consisting of 118 transitions, down from 427.

6 Experimental Results

In this section we report on the experiments we performed to compare FORT 2.0 with AGCP [1,2] and CoLL [8]. As starting point we consider the 121 left-linear right-ground TRSs in the latest version (765) of the Cops database.[3]

AGCP is a ground confluence tool for many-sorted TRSs based on rewriting induction. In Table 1 we compare FORT and AGCP v0.03 on one-sorted versions of the selected problems. Internally, FORT computes a compatible many-sorted signature with maximal number of sorts, when faced with a ground TRS. This is beneficial to reduce the set of possible ground terms, resulting in smaller automata. We used a 60 seconds time limit. FORT subsumes AGCP on our collection, with one exception. On Cops #741 AGCP reports "no" whereas FORT does not deliver an answer within 60 seconds. Increasing the time limit to 150 seconds enables FORT to report "no" as well.

CoLL is a confluence tool for left-linear TRSs based on commutation and it can establish commutation of multiple TRSs. In Table 2 we compare FORT 2.0 and the latest version[4] of CoLL on $7381 = \binom{121}{2} + 121$ commutation problems stemming from the selected 121 TRSs. To ensure compatibility of the signatures of the separate TRSs, we consistently renamed all function symbols (c_0, c_1, ... for constants, g_0, g_1, ... for unary symbols, etc.). Also for this comparison we used a 60 seconds time limit. FORT fully subsumes CoLL on our collection.

Detailed results can be obtained from the FORT website. AGCP and CoLL are not restricted to left-linear right-ground TRSs, but we believe that our research can help to make these (and other) tools stronger.

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

[4] Version 1.2 released on January 17, 2018; comparing FORT with version 1.1 of CoLL brought several bugs to light in the latter.

Table 1. AGCP versus FORT on 121 ground confluence problems.

tool	yes (∅ time)	no (∅ time)	maybe	timeout	total time
AGCP	24 (0.02 s)	78 (0.06 s)	15	4	276 s
FORT	37 (0.45 s)	81 (0.70 s)	–	3	253 s

Table 2. CoLL versus FORT on 7381 commutation problems.

tool	yes (∅ time)	no (∅ time)	maybe	timeout	total time
CoLL	623 (0.21 s)	–	276	6482	390682 s
FORT	761 (0.25 s)	6567 (0.60 s)	–	36	6308 s

Acknowledgments. We are grateful to Bertram Felgenhauer, Nao Hirokawa, and Julian Nagele for their support with the experiments. The comments by the anonymous reviewers helped to improve the presentation.

References

1. Aoto, T., Toyama, Y.: Ground confluence prover based on rewriting induction. In: Proc. 1st FSCD. LIPIcs, vol. 52, pp. 33:1–33:12 (2016), doi: 10.4230/LIPIcs.FSCD.2016.33
2. Aoto, T., Toyama, Y., Kimura, Y.: Improving rewriting induction approach for proving ground confluence. In: Proc. 2nd FSCD. LIPIcs, vol. 84, pp. 7:1–7:18 (2017), doi: 10.4230/LIPIcs.FSCD.2017.7
3. Dauchet, M., Tison, S.: The theory of ground rewrite systems is decidable. In: Proc. 5th LICS. pp. 242–248 (1990), doi: 10.1109/LICS.1990.113750
4. Durand, I., Middeldorp, A.: Decidable call-by-need computations in term rewriting. Information and Computation **196**(2), 95–126 (2005), doi: 10.1016/j.ic.2004.10.003
5. Middeldorp, A.: Approximating dependency graphs using tree automata techniques. In: Proc. 1st IJCAR. LNAI, vol. 2083, pp. 593–610 (2001), doi: 10.1007/3-540-45744-5_49
6. Rapp, F., Middeldorp, A.: Automating the first-order theory of left-linear right-ground term rewrite systems. In: Proc. 1st FSCD. LIPIcs, vol. 52, pp. 36:1–36:12 (2016), doi: 10.4230/LIPIcs.FSCD.2016.36
7. Rapp, F., Middeldorp, A.: Confluence properties on open terms in the first-order theory of rewriting. In: Proc. 5th IWC. pp. 26–30 (2016)
8. Shintani, K., Hirokawa, N.: CoLL: A confluence tool for left-linear term rewrite systems. In: Proc. 25th CADE. LNCS, vol. 9195, pp. 127–136 (2015), doi: 10.1007/978-3-319-21401-6_8
9. Sternagel, C., Sternagel, T.: Certifying confluence of almost orthogonal CTRSs via exact tree automata completion. In: Proc. 1st FSCD. LIPIcs, vol. 52, pp. 29:1–29:16 (2016), doi: 10.4230/LIPIcs.FSCD.2016.29
10. Zantema, H.: Automatically finding non-confluent examples in term rewriting. In: Proc. 2nd IWC. pp. 11–15 (2013)