Automating the First-Order Theory of Rewriting for Left-Linear Right-Ground Rewrite Systems*

Franziska Rapp[1] **and Aart Middeldorp**[2]

1 **Department of Computer Science, University of Innsbruck, Innsbruck, Austria**
franziska.rapp@uibk.ac.at
2 **Department of Computer Science, University of Innsbruck, Innsbruck, Austria**
aart.middeldorp@uibk.ac.at

Abstract

The first-order theory of rewriting is decidable for finite left-linear right-ground rewrite systems. We present a new tool that implements the decision procedure for this theory. It is based on tree automata techniques. The tool offers the possibility to synthesize rewrite systems that satisfy properties that are expressible in the first-order theory of rewriting.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases first-order theory, ground rewrite systems, automation, synthesis

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.36

1 Introduction

Dauchet and Tison [4] proved that the first-order theory of rewriting is decidable for ground rewrite systems. In this theory one can express properties like 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 normalization

$$\forall\, s\, \exists\, t\, (s \to^* t \wedge \neg \exists\, u\, (t \to u))$$

The decision procedure in [4] is based on tree automata techniques and easily extends to left-linear right-ground rewrite systems. Key ingredients are tree automata that accept encodings of n-ary relations on terms, ground tree transducers for certain binary relations on terms, as well as several closure properties corresponding to the logical operations of the first-order theory of rewriting.

We implemented the decision procedure in the new *First-Order Rewriting Tool*, FORT for short. It takes as input a left-linear right-ground TRS $\mathcal{R}$ and a first-order formula φ and reports whether $\mathcal{R}$ satisfies the property expressed by φ. We extended the theory with additional predicates in order to increase expressibility. For instance, the formula

$$\forall\, s\, \forall t\, \forall u\, (s \to^! t \wedge s \twoheadrightarrow u \implies t = u \vee \exists\, v\, (u \to^+ v \wedge v \xrightarrow{\epsilon} t))$$

can be handled by our tool. In addition, FORT can be used to synthesize left-linear right-ground rewrite systems that satisfy a property expressible in the first-order theory of rewriting, using a simple generate-and-test algorithm. Additional input parameters guide the search.

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

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 36; pp. 36:1–36:12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The remainder of this paper is organized as follows. In Section 2 the first-order theory of rewriting is formally defined and the supported predicates are presented. The tree automata techniques underlying the decision procedure implemented in FORT are covered in Section 3 and in Section 4 we explain how formulas are mapped to the constructions of the preceding section. Synthesis is the topic of Section 5. In Section 6 we briefly explain how FORT is used and present some implementation details. The power of FORT is illustrated by several experiments in Section 7. We conclude in Section 8 with related work and ideas for future work.

2 First-Order Theory of Rewriting

We assume familiarity with term rewriting [1]. We consider first-order logic over a language $\mathcal{L}$ without function symbols. The language contains the following binary predicate symbols:

$$\to \quad \to^+ \quad \to^* \quad \to^! \quad \between \quad \xrightarrow{\epsilon} \quad \xrightarrow{>\epsilon} \quad \leftrightarrow \quad \leftrightarrow^* \quad \downarrow \quad =$$

As models we consider finite TRSs $(\mathcal{F}, \mathcal{R})$ such that the set of ground terms $\mathcal{T}(\mathcal{F})$ is non-empty, which is equivalent to the requirement that the signature $\mathcal{F}$ contains at least one constant symbol. The set of ground terms serves as domain for the variables in formulas over $\mathcal{L}$. The interpretation of the predicate symbol $\to$ in $(\mathcal{F}, \mathcal{R})$ is the one-step rewrite relation $\to_\mathcal{R}$ over $\mathcal{T}(\mathcal{F})$. The other predicate symbols are interpreted in a similar fashion: $\to^!$ denotes normalization: $s \to^!_\mathcal{R} t$ if $s \to^*_\mathcal{R} t$ and t is a normal form of $\mathcal{R}$, $\between$ denotes a parallel rewrite step, $\xrightarrow{\epsilon}$ denotes one-step rewriting at the root position, $\xrightarrow{>\epsilon}$ denotes one-step rewriting below the root, and $\leftrightarrow^*$ denotes convertibility. The predicate symbols $\to^*$, $\to^!$, $\leftrightarrow$, and $\downarrow$ can be expressed in terms of the other predicate symbols:

$$s \to^* t \iff s \to^+ t \vee s = t \qquad s \leftrightarrow t \iff s \to t \vee t \to s$$
$$s \to^! t \iff s \to^* t \wedge \neg \exists u\, (t \to u) \qquad s \downarrow t \iff \exists u\, (s \to^* u \wedge t \to^* u)$$

We included them for convenience. For the same reason, $\mathcal{L}$ contains symbols denoting standard properties of (terms of) TRSs:

$$\mathsf{CR}(t) \iff \forall u \forall v\, (t \to^* u \wedge t \to v \implies u \downarrow v) \qquad \mathsf{CR} \iff \forall t\, \mathsf{CR}(t)$$
$$\mathsf{WCR}(t) \iff \forall u \forall v\, (t \to u \wedge t \to v \implies u \downarrow v) \qquad \mathsf{WCR} \iff \forall t\, \mathsf{WCR}(t)$$
$$\mathsf{WN}(t) \iff \exists u\, (t \to^! u) \qquad \mathsf{WN} \iff \forall t\, \mathsf{WN}(t)$$
$$\mathsf{UN}(t) \iff \forall u \forall v\, (t \to^! u \wedge t \to^! v \implies u = v) \qquad \mathsf{UN} \iff \forall t\, \mathsf{UN}(t)$$
$$\mathsf{NFP}(t) \iff \forall u \forall v\, (t \to u \wedge t \to^! v \implies u \to^! v) \qquad \mathsf{NFP} \iff \forall t\, \mathsf{NFP}(t)$$
$$\mathsf{NF}(t) \iff \neg \exists u\, (t \to u) \qquad \mathsf{UNC} \iff \forall t \forall u\, (t \leftrightarrow^* u \wedge \mathsf{NF}(t) \wedge \mathsf{NF}(u) \implies t = u)$$

The acronyms stand for the Church-Rosser property or confluence (CR), the weak Church-Rosser property or local confluence (WCR), (weak) normalization (WN), unique normal forms (UN), the normal form property (NFP), and unique normal forms with respect to conversion (UNC). On the other hand, the unary predicate symbol $\mathsf{Fin}_\circ$ defined as

$$\mathsf{Fin}_\circ(t) \iff \{u \mid t \circ u\} \text{ is finite}$$

for every boolean combination $\circ$ of the binary predicate symbols was added to $\mathcal{L}$ as it strictly increases expressibility. For example, it allows to express the important termination (strong normalization) property:

$$\mathsf{SN}(t) \iff \mathsf{Fin}_{\to^+}(t) \wedge \neg \exists u\, (t \to^* u \wedge u \to^+ u)$$
$$\mathsf{SN} \iff \forall t\, \mathsf{Fin}_{\to^+}(t) \wedge \neg \exists u\, (u \to^+ u)$$

It should be stressed that the above properties are restricted to *ground* terms. So CR stands for ground-confluence, which is different from confluence, even in the presence of ground terms; consider e.g. the TRS consisting of the rules $\mathsf{f}(x) \to x$ and $\mathsf{f}(x) \to \mathsf{c}$.

3 Tree Automata

Most of the results presented in this section are well-known. We refer to [3] for further details. We consider bottom-up tree automata $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ consisting of a finite signature $\mathcal{F}$ with $\mathcal{T}(\mathcal{F}) \neq \varnothing$, a finite set Q of states with $Q \cap \mathcal{F} = \varnothing$, a set $Q_f \subseteq Q$ of final states, and a set Δ of transition rules. Every transition rule has one of the following two shapes:

- $f(\alpha_1, \dots, \alpha_n) \to \beta$ with $f \in \mathcal{F}$ and $\alpha_1, \dots, \alpha_n, \beta \in Q$, or
- $\alpha \to \beta$ with $\alpha, \beta \in Q$.

Transition rules of the second shape are called ϵ-transitions. We view $\mathcal{A}$ as a ground TRS $(\mathcal{F} \cup Q, \Delta)$. The induced rewrite relation is denoted by $\to_{\mathcal{A}}$ or simply $\to$ if $\mathcal{A}$ can be inferred from the context. The language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is the set of ground terms in $\mathcal{T}(\mathcal{F})$ that can be rewritten to a final state. A subset $T \subseteq \mathcal{T}(\mathcal{F})$ is *regular* if there exists a tree automaton $\mathcal{A}$ with $L(\mathcal{A}) = T$.

Every regular language is accepted by a tree automaton $\mathcal{A}$ without ϵ-transitions. Moreover, it may be assumed that $\mathcal{A}$ is *deterministic* (no two different transition rules of $\mathcal{A}$ have the same left-hand side) and *completely defined* (for every $f(\alpha_1, \dots, \alpha_n)$ there exists a transition rule with this left-hand side).

▶ **Theorem 1.** *The class of regular languages is effectively closed under union, intersection, and complement.*

The constructions employed in the proof are well-known. In the decision procedure for the first-order theory of rewriting for left-linear right-ground TRSs they are used to encode the propositional connectives $\wedge$, $\vee$, and $\neg$.

▶ **Theorem 2.** *The following problems are decidable:*

instance:	*a tree automaton $\mathcal{A}$ and a ground term t*		
question:	$t \in L(\mathcal{A})$*?*		
instance:	*a tree automaton $\mathcal{A}$*	*instance:*	*tree automata $\mathcal{A}$ and $\mathcal{B}$*
question:	$L(\mathcal{A}) = \varnothing$*?*	*question:*	$L(\mathcal{A}) \subseteq L(\mathcal{B})$*?*

To cope with the predicate symbols in our language we use tree automata that operate on n-tuples of ground terms.

▶ **Definition 3.** For a signature $\mathcal{F}$ we let $\mathcal{F}^{(n)} = (\mathcal{F} \cup \{\bot\})^n$. Here, $\bot \notin \mathcal{F}$ is 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, …, f_n. Given terms $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})$, the term $\langle t_1, \dots, t_n \rangle \in \mathcal{T}(\mathcal{F}^{(n)})$ is defined inductively:

$$\langle t_1, \dots, t_n \rangle = \begin{cases} t_1 \cdots t_n & \text{if } t_1, \dots, t_n \text{ are constants} \\ f(u_1, \dots, u_m) & \text{otherwise} \end{cases}$$

where $f = \mathsf{root}(t_1) \cdots \mathsf{root}(t_n)$ has arity m and $u_i = \langle s_1, \dots, s_n \rangle$ with

$$s_j = \begin{cases} t_j|_i & \text{if } i \in \mathcal{P}\mathsf{os}(t_j) \\ \bot & \text{otherwise} \end{cases}$$

The following example illustrates the above definition.

▶ **Example 4.** Consider 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 $\langle s, t, u\rangle = \mathsf{fgf}(\mathsf{ggb}(\mathsf{aa}\bot), \mathsf{f}\bot\mathsf{g}(\mathsf{b}\bot\mathsf{a}, \mathsf{b}\bot\bot))$.

▶ **Definition 5.** A *regular relation* is an n-ary relation R on ground terms such that its encoding $\{\langle t_1, \ldots, t_n\rangle \mid (t_1, \ldots, t_n) \in R\}$ is regular. The class of all n-ary regular relations is denoted by RR_n.

An RR_1 relation is just a regular language. The following result is an immediate consequence of Theorem 1.

▶ **Theorem 6.** *The class of regular relations is effectively closed under boolean operations.*

▶ **Definition 7.** Let R be an n-ary relation over $\mathcal{T}(\mathcal{F})$.

- If $n \geqslant 2$ and $1 \leqslant i \leqslant n$ then the i-th *projection* $\Pi_i(R)$ of R is the relation

$$\{(t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n) \mid (t_1, \ldots, t_n) \in R\}$$

- If $1 \leqslant i \leqslant n+1$ then the i-th *cylindrification* $C_i(R)$ of R is the relation

$$\{(t_1, \ldots, t_{i-1}, u, t_i, \ldots, t_n) \mid (t_1, \ldots, t_n) \in R \text{ and } u \in \mathcal{T}(\mathcal{F})\}$$

- If σ is a permutation on $\{1, \ldots, n\}$ then $\sigma(R)$ denotes the relation

$$\{(t_{\sigma(1)}, \ldots, t_{\sigma(n)}) \mid (t_1, \ldots, t_n) \in R\}$$

▶ **Theorem 8.** *The class of regular relations is effectively closed under projection, cylindrification, and permutation.*

In the decision procedure projection is used to encode existential quantification, whereas cylindrification is used to combine relations of different arity. An important limitation of binary regular relations is that they are not closed under transitive closure [3, Exercise 3.1].

▶ **Definition 9.** A *ground tree transducer* (*GTT* for short) is a pair $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ of tree automata over the same signature $\mathcal{F}$. A pair of ground terms (s, t) is accepted by $\mathcal{G}$ if $s \to_{\mathcal{A}}^* \cdot \,{}_{\mathcal{B}}^*\!\leftarrow t$. The set of all accepted pairs is denoted by $R(\mathcal{G})$. A binary relation R on ground terms is a *GTT relation* if $R = R(\mathcal{G})$ for some GTT.

▶ **Theorem 10.** *The class of GTT relations is effectively closed under inverse and transitive closure.*

▶ **Theorem 11.** *If $\mathcal{G}_1 = (\mathcal{A}_1, \mathcal{B}_1)$ and $\mathcal{G}_2 = (\mathcal{A}_2, \mathcal{B}_2)$ are GTTs with disjoint sets of states and $\mathcal{G} = (\mathcal{A}_1 \cup \mathcal{A}_2, \mathcal{B}_1 \cup \mathcal{B}_2)$ then $R(\mathcal{G})^+ = (R(\mathcal{G}_1) \cup R(\mathcal{G}_2))^+$.*

▶ **Lemma 12.** *Every GTT relation is a regular relation.*

The final result in this section is an easy generalization of a result of Dauchet and Tison [4, 5]. The proof is given in the appendix.

▶ **Lemma 13.** *If R is a binary regular relation then Fin_R is a regular predicate.*

4 Automation

In this section we explain how first-order formulas are translated into tree automata manipulations. We start the discussion with the binary relation symbols of $\mathcal{L}$. Throughout this section, $\mathcal{R}$ denotes a finite left-linear right-ground TRS over a finite signature $\mathcal{F}$.

▶ **Lemma 14.** *The relations* $=$, $\to_{\mathcal{R}}$, $\xrightarrow{\epsilon}_{\mathcal{R}}$, *and* $\xrightarrow{>\epsilon}_{\mathcal{R}}$ *are regular.*

Rather than giving the proof, we illustrate the constructions on a simple example.

▶ **Example 15.** Consider the TRS $\mathcal{R}$ consisting of the rules $\mathsf{f}(x, \mathsf{a}) \to \mathsf{g}(\mathsf{b})$ and $\mathsf{g}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}, \mathsf{b})$. The (encoding of the) identity relation $=$ is accepted by the tree automaton

$$\mathsf{aa} \to \top \qquad \mathsf{bb} \to \top \qquad \mathsf{gg}(\top) \to \top \qquad \mathsf{ff}(\top, \top) \to \top$$

For $\xrightarrow{\epsilon}_{\mathcal{R}}$ we use the following tree automaton (with ✓ as unique final state):

$$\begin{array}{llll} \mathsf{fg}(\alpha, \beta) \to ✓ & \mathsf{ab} \to \alpha & \mathsf{a}\bot \to \gamma & \mathsf{gf}(\delta, \epsilon) \to ✓ \\ \mathsf{a}\bot \to \beta & \mathsf{bb} \to \alpha & \mathsf{b}\bot \to \gamma & \mathsf{aa} \to \delta \\ & \mathsf{gb}(\gamma) \to \alpha & \mathsf{g}\bot(\gamma) \to \gamma & \bot\mathsf{b} \to \epsilon \\ & \mathsf{fb}(\gamma, \gamma) \to \alpha & \mathsf{f}\bot(\gamma, \gamma) \to \gamma & \end{array}$$

By combining the previous two automata and adding the transitions

$$\mathsf{gg}(✓) \to ✓ \qquad \mathsf{ff}(✓, \top) \to ✓ \qquad \mathsf{ff}(\top, ✓) \to ✓$$

we obtain a tree automaton that accepts $\to_{\mathcal{R}}$. For $\xrightarrow{>\epsilon}_{\mathcal{R}}$ we add instead

$$\begin{array}{lll} \mathsf{gg}(✓) \to ✓' & \mathsf{ff}(✓, \top) \to ✓' & \mathsf{ff}(\top, ✓) \to ✓' \\ \mathsf{gg}(✓') \to ✓' & \mathsf{ff}(✓', \top) \to ✓' & \mathsf{ff}(\top, ✓') \to ✓' \end{array}$$

and make ✓′ the unique final state.

▶ **Lemma 16.** *The relation* $\twoheadrightarrow_{\mathcal{R}}$ *is a GTT relation.*

▶ **Example 17.** The GTT $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ with

$$\mathcal{A} \quad \begin{array}{llll} \mathsf{a} \to \alpha & \mathsf{g}(\alpha) \to \alpha & \mathsf{a} \to \beta & \mathsf{f}(\alpha, \beta) \to \iota_1 \\ \mathsf{b} \to \alpha & \mathsf{f}(\alpha, \alpha) \to \alpha & & \mathsf{g}(\beta) \to \iota_2 \end{array} \quad \Big| \quad \begin{array}{ll} \mathsf{g}(\gamma) \to \iota_1 & \mathsf{a} \to \delta \\ \mathsf{f}(\delta, \gamma) \to \iota_2 & \mathsf{b} \to \gamma \end{array} \quad \mathcal{B}$$

accepts the relation $\twoheadrightarrow_{\mathcal{R}}$ for the TRS $\mathcal{R}$ of the previous example.

▶ **Lemma 18.** *The relations* $\to^*_{\mathcal{R}}$, $\to^+_{\mathcal{R}}$, *and* $\leftrightarrow^*_{\mathcal{R}}$ *are regular relations.*

Proof. The relation $\to^*_{\mathcal{R}} = \twoheadrightarrow^+_{\mathcal{R}}$ is a GTT relation according to Lemma 16 and Theorem 10, and hence regular according to Lemma 12. To obtain the regularity of $\to^+_{\mathcal{R}}$, the construction in the proof of Lemma 12 [3, Proposition 3.2.7] is slightly modified. For $\leftrightarrow^*_R = (\twoheadrightarrow_{\mathcal{R}} \cup {}_{\mathcal{R}}\twoheadleftarrow)^+$ we additionally use Theorem 11. ◀

Note that we cannot use the equivalence of $\leftrightarrow^*_{\mathcal{R}}$ and $(\to_{\mathcal{R}} \cup {}_{\mathcal{R}}\leftarrow)^+ \cup =$ in the above proof since $\to_{\mathcal{R}} \cup {}_{\mathcal{R}}\leftarrow$ is not a GTT relation and binary regular relations are not closed under transitive closure.

For the remaining binary relations we use the formulas given in Section 2 in combination with the closure properties of Section 3. We have to be careful when combining relations with union and intersection because there may be a mismatch in the dimension of the involved relations. Consider the binary relations $R_1 = \{(s, t) \mid s \to t\}$ and $R_2 = \{(t, u) \mid t \to u\}$. Their union is the ternary relation $R_1 \cup R_2 = \{(s, t, u) \mid s \to t \text{ or } t \to u\}$, so before applying the union construction we have to cylindrify R_1 and R_2 suitably. Also permutation is needed if the order of terms in the participating relations is different.

Existential quantification is implemented using projection (Definition 7 and Theorem 8). Formulas are normalized in such a way that implications and universal quantifications are eliminated using the known equivalences ($\varphi \Rightarrow \psi \equiv \neg\,\varphi \vee \psi$ and $\forall\, x\, \varphi \equiv \neg\,\exists\, x\, \neg\varphi$). Furthermore, negations are pushed inside and double negations are eliminated. We do not transform formulas into prenex normal form since this typically increases the dimension of the relations, with a corresponding increase in the computation time for the closure operations.

We illustrate the decision procedure on a small example.

▶ **Example 19.** Suppose we want to determine whether a given left-linear right-ground TRS $\mathcal{R}$ is (weakly) normalizing. We use the formula $\phi = \forall\, s\, \exists\, t\, (s \to^* t \wedge \neg\,\exists\, u\, (t \to u))$ for this purpose. The computed (tree automata representations of the) relations are listed below:

1. GTT relation for $\{(s,t) \mid s \twoheadrightarrow_{\mathcal{R}} t\}$ (Lemma 16)
2. GTT relation for $\{(s,t) \mid s \to^*_{\mathcal{R}} t\}$ (Theorem 10)
3. RR_2 relation for $\{(s,t) \mid s \to^*_{\mathcal{R}} t\}$ (Lemma 12)
4. RR_2 relation for $\{(t,u) \mid t \to_{\mathcal{R}} u\}$ (Lemma 14)
5. RR_1 relation for $\{t \mid \exists\, u\, (t \to_{\mathcal{R}} u)\}$ (projection Π_2)
6. RR_1 relation for $\{t \mid \neg\,\exists\, u\, (t \to_{\mathcal{R}} u)\}$ (complement)
7. RR_2 relation for $\{(s,t) \mid \neg\,\exists\, u\, (t \to_{\mathcal{R}} u)\}$ (cylindrification C_1)
8. RR_2 relation for $\{(s,t) \mid s \to^*_{\mathcal{R}} t \wedge \neg\,\exists\, u\, (t \to_{\mathcal{R}} u))\}$ (intersection)
9. RR_1 relation for $\{s \mid \exists\, t\, (s \to^*_{\mathcal{R}} t \wedge \neg\,\exists\, u\, (t \to_{\mathcal{R}} u))\}$ (projection Π_2)
10. RR_1 relation for $\{s \mid \neg\,\exists\, t\, (s \to^*_{\mathcal{R}} t \wedge \neg\,\exists\, u\, (t \to_{\mathcal{R}} u))\}$ (complement)

Then $\mathcal{R}$ satisfies ϕ if and only if the final language is empty (Theorem 2).

The logic supports the shorter formula $\forall\, s\, \exists\, t\, (s \to^! t)$ to test for normalization. The relation $\{(s,t) \mid s \to^! t\}$ is implemented as $\{(s,t) \mid s \to^* t \wedge \mathsf{NF}(t)\}$. For the NF predicate our tool FORT employs the explicit construction presented in Comon [2, Section 4.2] rather than the implicit encoding $\neg\,\exists\, u\, (t \to u)$ given in the preceding example. For small TRSs there is little difference, but if the left-hand sides have a larger depth or contain functions symbols of a higher arity, the explicit construction is more efficient.

5 Synthesis

Our tool can also be used to synthesize TRSs that satisfy properties given by the user. This may be interesting for finding counterexamples and non-trivial TRSs for exam exercises as well as competitions. Furthermore, we envisage the use of FORT in courses on term rewriting.

The synthesis algorithm for *ground* TRSs is shown in Figure 1. Of the five input parameters, the formula ϕ must be supplied whereas the other four are optional. If the signature $\mathcal{F}$ is not given, we enumerate all finite signatures in a dovetailing manner and call the procedure repeatedly. The default values for the other optional parameters are $n = 3$, $m = 4$, and $d = m - 1$. The outer while loop incrementally constructs the signature $\mathcal{F}$. The current candidate $\mathcal{G} \subseteq \mathcal{F}$ is used to compute the set of terms T from which the rewrite rules are constructed. Terms in T must satisfy the given depth and size restrictions. In the for loop we enumerate all candidate ground TRSs that have no more than n rules, starting from the ones with the fewest rules. In the inner while loop we consider the set of ground TRSs R' consisting of i rewrite rules. We select the smallest $\mathcal{R} \in R'$ in some fixed total order $>$ on TRSs and test whether $\mathcal{R}$ is *normalized*, which means in this context that no smaller

Input: • closed first-order formula ϕ over $\mathcal{L}$
• signature $\mathcal{F}$ • maximum size of terms m
• maximum number of rules n • maximum depth of terms d

Output: • TRS $\mathcal{R}$ satisfying ϕ

$\mathcal{G} := \varnothing$;
while $\mathcal{F} \setminus \mathcal{G} \neq \varnothing$ **do**
  $\mathcal{G} := \mathcal{G} \cup \{(f, a)\}$ with $(f, a) \in \mathcal{F} \setminus \mathcal{G}$ of minimum arity a;
  $T := \{t \in \mathcal{T}(\mathcal{G}) \mid |t| \leqslant m \text{ and } \mathsf{depth}(t) \leqslant d\}$;
  $R := T \times T$;
  for $i = 0$ **to** $\min(n, |R|)$ **do**
    $R' := \{S \subseteq R \mid S \text{ consists of } i \text{ pairs}\}$;
    while $R' \neq \varnothing$ **do**
      select smallest $\mathcal{R} \in R'$ (with respect to some total order $>$);
      if $\mathcal{R}$ is normalized and $\mathcal{R} \vDash \phi$ **then return** $\mathcal{R}$;
      $R' := R' \setminus \{\mathcal{R}\}$
    od
  od
od;
return *failure*

Figure 1 Procedure for synthesizing TRSs.

(with respect to $>$) TRS exists (in the original R') that can be obtained from $\mathcal{R}$ by renaming constant symbols (within $\mathcal{G}$). This test is done to avoid calling the decision tool on a TRS for which an earlier call on a renamed version of the TRS was unsuccessful. If $\mathcal{R}'$ passes this test we use the decision tool to determine whether $\mathcal{R}$ satisfies the formula ϕ.

The procedure is easily extended to left-linear right-ground TRSs. We have an additional optional input parameter v for the number of variables. Its default value is one. A set T' of linear terms with variables sorted in some fixed order (to keep the set small) is computed and the set of candidate TRSs is extended to $T' \times T$.

6 Interface

Precompiled binaries to run FORT from the command line are available from

`http://cl-informatik.uibk.ac.at/software/FORT`

FORT can be used as decision tool via the following command

```
./fort -D <file> [<verbosity>] "<formula>"
```

where `<formula>` specifies the property that should be tested on the TRS given in `<file>`. The syntax of TRSs follows the standard TPDB format.[1] The optional `<verbosity>`

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

parameter takes a value in $\{0, 1, 2\}$. The higher the value, the more detailed the output will be. The default value is 0. The syntax of the formula is explained in the file `description.txt`. The quotation marks around the formula are essential. The basic command to synthesize TRSs is

```
./fort -S <file>
```

where <file> must contain the line

```
(FORMULA <formula>)
```

and may contain the lines

```
(SIGNATURE <F>)  (VARIABLES <V>)  (RULES <R>)  (SIZE <S>)  (DEPTH <D>)
```

corresponding to the optional parameters that were described in the preceding section. The signature `<F>` is specified as a comma separated list of function symbols and arities. For example, `(a 0, g 1, h 1)` specifies a constant `a` and two unary symbols `g` and `h`. The information in `<file>` can also be passed to FORT via the command line options

```
-f "<F>"        -v <V>        -r <R>        -s <S>        -d <D>
```

7 Experiments

In this section we illustrate FORT on a number of examples. After reporting on some experiments we performed with the decision tool, we turn to the synthesis part.

The combined confluence[2] and termination[3] problem databases contain 65 left-linear right-ground TRSs. When testing for confluence, FORT reports that 42 of these TRSs are (ground-)confluent, 14 are non-(ground-)confluent, and the remaining 9 TRSs exceed the given time limit of 60 seconds. (Ground-confluence coincides with confluence on our collection of TRSs.) Not surprisingly, designated confluence provers[4] like ACP, CSI, and Saigawa are considerably faster than FORT, but only CSI subsumes FORT on the 65 left-linear right-ground TRSs (45/19/1 in less than 2 minutes, compared to the 8 minutes of FORT). Next we compare the following three different but equivalent formulations of confluence:

`forall s, t, u (s ->* t & s ->* u => t join u)`	$631s$, 10 timeouts	(1)
`forall s, t, u (s ->* t & s -> u => t join u)`	$484s$, 9 timeouts	(2)
`forall t, u (t <->* u => t join u)`	$895s$, 13 timeouts	(3)

The total time spent by FORT (using a 60 seconds time limit) is given in the right column. The computation of $\leftrightarrow^*$ is much more expensive than $\rightarrow^*$. Moreover, intersecting the RR_3 relations $R_1 = \{(s,t,u) \mid s \rightarrow^* t\}$ and $R_2 = \{(s,t,u) \mid s \rightarrow^* u\}$ is more expensive than intersecting R_1 with $R_3 = \{(s,t,u) \mid s \rightarrow u\}$. The numbers explain why the second formula (called *semi-confluence* in [1]) is selected as translation of the predicate CR.

[2] `http://cops.uibk.ac.at/`
[3] `http://termination-portal.org/wiki/TPDB`
[4] `http://coco.nue.riec.tohoku.ac.jp/tools/`

Table 1 Intermediate automata sizes for TRS #74 of the confluence database.

property	$\to$	$\to^*$	$\leftrightarrow^*$	$\cap_1$	$\cap_2$	$\cap_3$	$\cap_4$	time
(1)		82		1162	2184	33088	1723	$5.2s$
(2)	37	82		472	2184	15187	910	$2.1s$
(3)		82	1634		2184	43076	19237	$27.1s$

To give an idea of the sizes of the computed tree automata, we present detailed information in Table 1 for TRS #74

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

$\mathsf{b} \to \mathsf{c}$ $\quad$ $\mathsf{f}(\mathsf{c}, x) \to \mathsf{f}(\mathsf{c}, \mathsf{c})$ $\quad$ $\mathsf{d} \to \mathsf{f}(\mathsf{c}, \mathsf{b})$

of the confluence database. Here $\cap_1$ corresponds to the conjunction in the subformula that represents the peak (absent for property (3)) and $\cap_2$ the one from the join $t \downarrow u$. The next two columns correspond to the implication in the three properties before ($\cap_3$) and after ($\cap_4$) trimming the automata. As can be seen, trimming intermediate automata, although time consuming, has a significant impact.

We performed a similar experiment for the normal form property (NFP):

`forall s, t, u (s -> t & s ->! u => t ->* u)`	$187s$, 4 timeouts
`forall t, u (t <->* u & NF(u) => t ->* u)`	$863s$, 13 timeouts
`forall t (WN(t) => CR(t))`	$496s$, 7 timeouts

justifying the selection of the first formula.

The time required to synthesize a TRS varies vastly, depending on the property, the additional options, and the size of the smallest TRS fulfilling the property. Hence, the options should be chosen with care.

▶ **Example 20.** We start with looking for non-confluent TRSs having unique normal forms (`UN & ~CR`). With the option `(VARIABLES 0)` the ground TRS

$\mathsf{a} \to \mathsf{a}$ $\quad$ $\mathsf{b} \to \mathsf{a}$ $\quad$ $\mathsf{b} \to \mathsf{g}(\mathsf{a})$

is generated in about 18 seconds. Generating a one-rule TRS with the option `(RULES 1)` produces $\mathsf{g}(\mathsf{g}(x)) \to \mathsf{h}(\mathsf{g}(\mathsf{a}))$ in a bit more than 2 seconds.

▶ **Example 21.** Next we consider the formula

$\neg \forall t\ \mathsf{Fin}_{\overleftarrow{\epsilon}}(t)$ $\quad$ `(FORMULA ~forall t Fin(e<-,t))`

which distinguishes ground TRSs from left-linear right-ground (but not ground) ones. Without any options FORT produces the single rule $\mathsf{g}(x) \to \mathsf{c}$ in a fraction of a second. The formula

$\neg \forall t\ \mathsf{Fin}_{\neq}(t)$ $\quad$ `(FORMULA ~forall t Fin(~=,t))`

is true for TRSs that are not ARSs. FORT produces the empty TRS over the signature consisting of two constants and a unary function symbol. To ensure the existence of a function symbol of arity $n > 1$ we can use the formula $\exists s\, \exists t\, (s \twoheadrightarrow t \wedge \neg(s \to t) \wedge \neg(s = t))$, which results in the single rule $\mathsf{a} \to \mathsf{b}$ over a signature containing an additional binary function symbol f.

▶ **Example 22.** Finding a locally confluent but not confluent TRS $\mathcal{R}$ is easy. FORT produces the two-rule TRS

$$\mathsf{g}(\mathsf{g}(\mathsf{c})) \to \mathsf{c} \qquad\qquad \mathsf{g}(\mathsf{g}(\mathsf{c})) \to \mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{c})))$$

when giving the formula `(WCR & ~CR)`. The well-known abstract counterexample by Kleene

$$\mathsf{a} \longleftarrow \mathsf{b} \rightleftarrows \mathsf{c} \longrightarrow \mathsf{d}$$

is found by restricting the search to ARSs. This can be done either by extending the formula to `(WCR & ~CR & forall t Fin(~=,t))` or by using the option `(DEPTH 0)`. Moreover, the default value for the maximal number of rewrite rules has to be increased to at least four `(RULES 4)`. If we impose the condition that $\mathcal{R}^{-1}$ is terminating (cf. [11]), the TRS

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

is produced with `(WCR & ~CR & forall t Fin(<-,t) & ~exists u (u +<- u))`.

8 Conclusion

Concerning related work, we are not aware of any other tree-automata based tool for synthesizing TRSs nor of any tool that allows properties to be specified by an arbitrary first-order formula in the theory of rewriting. Jiresch [6] developed a synthesis tool to attack the well-known open problems (RTA LOOP #13) concerning the sufficiency of certain restricted joinability conditions on critical pairs of left-linear TRSs.

Zantema [11] developed the tool Carpa+ for synthesizing TRSs that satisfy properties which can be encoded as SMT problems. The TRSs that can be synthesized form a small extension of the class of ARSs: A single unary function symbol f is permitted and rules must have the form $a \to b$, $a \to f(b)$, or $f(a) \to b$, where a and b are constants. The properties are restricted to those that can be encoded into the conjunctive fragment of SMT-LRA (linear real arithmetic). The predecessor tool Carpa synthesized combinations of ARSs with help of a SAT solver. It was used to show the necessity of certain conditions in abstract confluence results [8, Section 5].

We conclude by mentioning some ideas for future work. The most interesting extension is the generation of witnesses for existential formulas or formulas with free variables. The efficiency of FORT can certainly be improved, e.g. by converting its sequential code into multi-threaded code. Optimizing the synthesis algorithm for ARSs is easily possible. Support for combinations of TRSs (e.g., to express commutation) is also useful. The enumeration algorithm can be improved, for instance by having the values that restrict the search space increase over time. The question whether rewrite strategies can be incorporated is less easy to answer. A major obstacle is that the subterm relation is not expressible in the first-order theory of rewriting (cf. [3, Exercise 3.13]). Going beyond left-linear right-ground TRSs is a non-trivial matter. Dropping either restriction, one quickly faces an undecidable first-order theory, even when one-step rewriting ($\to$) is the only predicate symbol [9, 7, 10]. Formalizing the underlying theory into an interactive theorem prover like Isabelle/HOL will be a major undertaking, but a necessary step to ensure that the answers and TRSs produced by FORT are correct.

Acknowledgements. We thank Bertram Felgenhauer for reporting a mistake in the earlier implementation of projection.

References

1 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

2 H. Comon. Sequentiality, monadic second-order logic and tree automata. *I&C*, 157(1-2):25–51, 2000. doi:10.1006/inco.1999.2838.

3 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2007. URL: http://www.grappa.univ-lille3.fr/tata.

4 M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th LICS*, pages 242–248, 1990. doi:10.1109/LICS.1990.113750.

5 M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable (extended version). Technical Report I.T. 197, LIFL, 1990.

6 E. Jiresch. A term rewriting laboratory with systematic and random generation and heuristic test facilities. Master's thesis, Vienna University of Technology, 2008. URL: http://www.logic.at/staff/jiresch/thesis/thesis.pdf.

7 J. Marcinkowski. Undecidability of the first order theory of one-step right ground rewriting. In *Proc. 8th RTA*, volume 1232 of *LNCS*, pages 241–253, 1997. doi:10.1007/3-540-62950-5_75.

8 A. Stump, H. Zantema, G. Kimmell, and R. El Haj Omar. A rewriting view of simple typing. *LMCS*, 9(1), 2012. doi:10.2168/LMCS-9(1:4)2013.

9 R. Treinen. The first-order theory of linear one-step rewriting is undecidable. *TCS*, 208(1-2):179–190, 1998. doi:10.1016/S0304-3975(98)00083-8.

10 S. Vorobyov. The undecidability of the first-order theories of one step rewriting in linear canonical systems. *I&C*, 175(2):182–213, 2002. doi:10.1006/inco.2002.3151.

11 H. Zantema. Automatically finding non-confluent examples in term rewriting. In *Proc. 2nd IWC*, pages 11–15, 2013. URL: http://cl-informatik.uibk.ac.at/iwc/iwc2013.pdf.

A Proof of Lemma 13

Proof. The proof is based on the sketch in Dauchet and Tison [5]. Let $\mathcal{A} = (\mathcal{F}^{(2)}, Q, Q_f, \Delta)$ be the tree automaton that accepts the binary regular relation R. The set Q_∞ is defined as follows:

$$Q_\infty = \{q \in Q \mid \langle \bot, t\rangle \to^*_{\mathcal{A}} q \text{ for infinitely many terms } t \in \mathcal{T}(\mathcal{F})\}$$

The set Q_∞ is computed by constructing a graph with states as nodes. For every transition rule $\bot f(q_1, \dots, q_n) \to q$ in Δ we add transitions from $q_1, \dots, q_n$ to q. Now, $q \in Q_\infty$ if there exists a cycle from which q can be reached.

Next we define an automaton $\mathcal{A}' = (\mathcal{F}, Q \cup \bar{Q}, \bar{Q}_f, \Delta \cup \Delta')$. Here, $\bar{Q}$ is a copy of Q where every state is dashed: $q \in \bar{Q}$ if and only if $q \in Q$. For every transition rule $fg(q_1, \dots, q_n) \to q \in \Delta$ we have the following rules in Δ':

$$fg(q_1, \dots, q_n) \to \bar{q} \qquad \text{if } q_i \in Q_\infty \text{ for some } i > \operatorname{arity}(f) \tag{1}$$

$$fg(q_1, \dots, \bar{q}_i, \dots, q_n) \to \bar{q} \qquad \text{for all } 1 \leqslant i \leqslant n \tag{2}$$

Finally we define the automaton $\mathcal{B} = (\mathcal{F}, Q_\mathcal{B}, Q_{\mathcal{B}f}, \Delta_B)$ as the complement of the second projection $\Pi_2(R(\mathcal{A}'))$ of $\mathcal{A}'$. Since the construction for projection does not change (final) states, we have $Q_\mathcal{B} = Q \cup \bar{Q}$ and $Q_{\mathcal{B}f} = Q_\mathcal{B} \setminus \bar{Q}_f$. We show $L(\mathcal{B}) = \{t \in \mathcal{T}(\mathcal{F}) \mid \mathsf{Fin}_R(t)\}$.

$\subseteq$ Let $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F})$ and assume that $\mathsf{Fin}_R(t)$ does not hold, i.e., the set $\mathcal{U} = \{u \in \mathcal{T}(\mathcal{F}) \mid (t, u) \in R\}$ is infinite. Since the signature $\mathcal{F}$ is finite, infinitely many terms u in $\mathcal{U}$ have a depth greater than t. Hence there exists a position $p \notin \mathcal{P}\mathsf{os}(t)$ such that the set $\mathcal{U}' = \{u \in \mathcal{U} \mid p \in \mathcal{P}\mathsf{os}(u)\}$ is infinite. For every $u \in \mathcal{U}'$ we have $\langle t, u\rangle|_p = \langle \bot, u|_p\rangle$. Since $\langle t, u\rangle$ is accepted by $\mathcal{A}$ and Q is finite, there must exist a state q' such that $\langle \bot, u|_p\rangle \to^*_{\mathcal{A}} q'$ for infinitely many terms $u \in \mathcal{U}'$. Therefore $q' \in Q_\infty$. By construction of $\mathcal{A}'$, there must be a transition rule $f'g(q_1, \ldots, q_m) \to \bar{q}$ in Δ' of type (1) such that $q_i = q'$ for some $i > \mathsf{arity}(f')$. Pick any $u \in \mathcal{U}'$ such that $\langle \bot, u|_p\rangle \to^*_{\mathcal{A}} q'$. We distinguish two cases, depending on the position p.

- If $u|_p$ is a direct subterm of u then $f = f'$, $i = p$, and $\bar{q} \in \bar{Q}_f$ by construction of $\mathcal{A}'$. Note that $p > \mathsf{arity}(f)$ because $p \notin \mathcal{P}\mathsf{os}(t)$. Since $(t, u) \in R(\mathcal{A}')$, $t \in \Pi_2(R(\mathcal{A}'))$ and thus $t \notin L(\mathcal{B})$ according to the definition of $\mathcal{B}$.
- Otherwise, the dash of $\bar{q}$ is propagated upwards using transition rules of type (2), such that we obtain $\langle t, u\rangle \to^*_{\mathcal{A}'} \bar{q}_f$ for some $\bar{q}_f \in \bar{Q}_f$. Hence $(t, u) \in R(\mathcal{A}')$ and we complete the proof as in the previous case.

$\supseteq$ Let $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F})$ and assume $t \notin L(\mathcal{B})$. By construction of $\mathcal{B}$ there exists a term $u = g(u_1, \ldots, u_m) \in \mathcal{T}(\mathcal{F})$ such that $\langle t, u\rangle \in L(\mathcal{A}')$. Hence there exists a transition rule $fg(q_1, \ldots, q_k) \to \bar{q}_f$ in Δ' with $k = \max(n, m)$ and $\bar{q}_f \in \bar{Q}_f$. We distinguish two cases, depending on the transition rule.

- Suppose the transition rule is of type (2). Hence there exists a state $q_i \in \bar{Q}$ with $1 \leqslant i \leqslant n$ such that $\langle t_i, u_i\rangle \to^*_{\mathcal{A}'} q_i$ and there must be a position $p \in \mathcal{P}\mathsf{os}(u_i) \setminus \mathcal{P}\mathsf{os}(t_i)$ such that $\langle \bot, u_i|_p\rangle \to^*_{\mathcal{A}'} q$ for some state $q \in Q_\infty$. By definition of Q_∞, $\langle \bot, s\rangle \to^*_{\mathcal{A}'} q$ and thus also $\langle t, u[s]_{ip}\rangle \to^*_{\mathcal{A}'} \bar{q}_f$ for infinitely many terms $s \in \mathcal{T}(\mathcal{F})$. Hence the set $\{u \mid (t, u) \in R\}$ is infinite and therefore $\mathsf{Fin}_R(t)$ does not hold.
- Suppose the transition rule is of type (1). So $q_i \in Q_\infty$ for some $i > n$ and thus $m > n$. Hence $\langle \bot, u_i\rangle \to^*_{\mathcal{A}'} q_i$ and there exist infinitely many other terms s such that $\langle \bot, s\rangle \to^*_{\mathcal{A}'} q_i$ and $\langle t, u[s]_i\rangle \to^*_{\mathcal{A}'} \bar{q}_f$. Hence the set $\{u \mid (t, u) \in R\}$ is infinite and $\mathsf{Fin}_R(t)$ does not hold as before. ◀