



First-Order Theory of Rewriting for Linear Variable-Separated Rewrite Systems: Automation, Formalization, Certification

Aart Middeldorp¹ · Alexander Lochmann¹ · Fabian Mitterwallner¹

Received: 22 April 2022 / Accepted: 19 January 2023
© The Author(s) 2023

Abstract

The first-order theory of rewriting is decidable for linear variable-separated rewrite systems. We present a new decision procedure which is the basis of FORT, a decision and synthesis tool for properties expressible in the theory. The decision procedure is based on tree automata techniques and verified in Isabelle. Several extensions make the theory more expressive and FORT more versatile. We present a certificate language that enables the output of FORT to be certified by the certifier FORTify generated from the formalization, and we provide extensive experiments.

Keywords Term rewriting · First-order theory · Tree automata · Formalization

1 Introduction

Many properties of rewrite systems can be expressed as logical formulas in the first-order theory of rewriting. This theory is decidable for the class of linear variable-separated rewrite systems, which includes all ground rewrite systems. The decision procedure is based on tree automata techniques and goes back to Dauchet and Tison [10]. It is implemented in FORT [46, 48], which takes as input one or more rewrite systems $\mathcal{R}_0, \mathcal{R}_1, \dots$ and a formula φ , and determines whether the rewrite systems satisfy the property expressed by φ , in which case it reports yes or no. FORT may not reach a conclusion due to limited resources.

For properties related to confluence and termination, designated competitions (CoCo [41], termCOMP [23]) of software tools take place regularly. Occasionally, yes/no conflicts appear. Since the participating tools typically couple a plethora of techniques with sophisticated search strategies, human inspection of the output of tools to determine the correct answer is often not feasible. Hence certified categories were created in which tools must output a formal certificate. This certificate is verified by CeTA [53], an automatically generated Haskell program using the code generation feature of Isabelle. This requires not only that the underlying techniques are formalized in Isabelle, but the formalization must be executable for code generation to apply. During the time-consuming formalization process, mistakes in

✉ Aart Middeldorp
aart.middeldorp@uibk.ac.at

¹ Department of Computer Science, University of Innsbruck, Innsbruck, Austria

papers are sometimes brought to light. An additional outcome is that formalization efforts may give rise to simpler and more efficient constructions and algorithms.

Since 2017 we are concerned with the question of how to ensure the correctness of the answers produced by FORT. The certifier CeTA supports a great many techniques for establishing concrete properties like termination and confluence, but the formalizations in the underlying Isabelle Formalization of Rewriting (IsaFoR)¹ are orthogonal to the ones required for supporting the decision procedure underlying FORT. We present a certificate language which is rich enough to express the various automata operations in decision procedures for the first-order theory of rewriting as well as numerous predicate symbols that may appear in formulas in this theory. FORTify, the verified Haskell program obtained from the Isabelle formalization, validates certificates in this language.

The decision procedure implemented in FORT and formalized in Isabelle is based on three different tree automata models. We use standard bottom-up tree automata to represent various sets of ground terms. For (most) binary relations on ground terms, we use *anchored* ground tree transducers. These are a simplification of the ground tree transducers used in the literature [8–10, 12, 18] with better closure properties, reducing the number of constructions needed to represent the first-order theory of rewriting. Some of these closure properties are proved (and formalized) using the simple but equivalent class of *pair automata*. The third model are standard tree automata operating on a different signature in order to represent n -ary relations on ground terms, for arbitrary n (including $n = 2$). In the next section we present the basic definitions. Section 3 introduces the first-order theory of rewriting. In Sect. 4 we introduce in a systematic way several context closure operations on binary relations that are used to represent the binary predicates in the first-order theory of rewriting. Detailed proofs of the various results concerning the three tree automata models that are required for the decision procedure are presented in Sect. 5. Many of the results and tree automata constructions in this section are well-known, but are included for completeness and because the implementation in FORT and the subsequent formalization are directly based on them. Tree automata operate on ground terms. In Sect. 6 we present the formalized signature extension results that allow to reduce certain properties on arbitrary terms to properties on ground terms. In Sect. 7 the decision and synthesis modes of FORT are described, and a new undecidability proof related to the latter is presented. We also discuss the representation of formulas in certificates and the certificate language, and we explain how certificates are validated by FORTify, the verified Haskell program obtained from the Isabelle formalization. Experimental results are presented in Sect. 8, before we conclude in Sect. 9. In an appendix the input syntax and the interface of the tools is presented.

The formalization is based on Isabelle/HOL. Our contribution is split into three parts, which are published as separate entries in the Archive of Formal Proofs.² The first part [35] contains general results about bottom-up tree automata, ported from IsaFoR, extended with constructions and results about anchored ground tree transducers, pair automata, and regular relation automata. The second part [33] formalizes primitive constructions needed to decide the first-order theory of rewriting. Moreover, it connects the logical semantic entailment of first-order formulas to regular tree languages. This connection gives rise to a natural description of the decision procedure. The specification allows tool authors to generate certificates (which can be viewed as a formal proof claim using appropriate automata construction for the corresponding logical connectives and predicates). We rely on the code generation facility


¹ <http://cl-informatik.uibk.ac.at/isafor/>

² <https://www.isa-afp.org>

of Isabelle/HOL to obtain the certifier FORTify that is able to verify the integrity of such certificates. The third part [32] is independent, and covers the results in Sect. 6.

The formalization can be accessed via the following links:

- https://www.isa-afp.org/entries/Regular_Tree_Relations.html
- https://www.isa-afp.org/entries/FO_Theory_Rewriting.html
- https://www.isa-afp.org/entries/Rewrite_Properties_Reduction.html

Most definitions, theorems, and lemmata in this paper directly correspond to the formalization. These are indicated by the  symbol, which links to an HTML rendering of our formalization, for those who like to dive right into the actual Isabelle code. In the running text (traditional) proof details are given.

This article combines and extends earlier papers that appeared in conference and informal workshop proceedings. These cover system descriptions of earlier versions of FORT [46, 48], formalization and certification aspects [22, 34, 36, 42], as well as results for dealing with properties on non-ground terms [37, 38, 47]. Many new examples to illustrate the various constructions were added and the presentation is self-contained. The efficiency improvements described in Sect. 7 are new. The same is true for the undecidability result in Sect. 7.5. Also several of the experiments that we present in Sect. 8 have not been described before.

2 Preliminaries

In this preliminary section we recall basic definitions and notations of term rewriting [3] and tree automata [8].

2.1 Term Rewriting

We assume a finite signature \mathcal{F} containing at least one constant symbol and a disjoint set of variables \mathcal{V} . The set of terms built up from \mathcal{F} and \mathcal{V} is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$, while $\mathcal{T}(\mathcal{F})$ denotes the (non-empty) set of ground terms. The set of variables occurring in a term t is denoted by $\text{Var}(t)$. A term is linear if it does not contain multiple occurrences of the same variable. Positions are strings of positive integers which are used to address subterms. The set of positions in a term t is denoted by $\text{Pos}(t)$ and the root position by ε . The function symbol at position $p \in \text{Pos}(t)$ is denoted by $t(p)$ and $t[u]_p$ denotes the result of replacing the subterm $t|_p$ of t at position p by the term u . The height $\text{height}(t)$ of a term t is the length of a longest position in $\text{Pos}(t)$. A substitution is a mapping σ from variables to terms and $t\sigma$ denotes the result of applying σ to a term t . A context C is a term that contains exactly one hole, denoted by the special constant $\square \notin \mathcal{F}$. We write $C[t]$ for the result of replacing the hole in C by the term t . A term rewrite system (TRS) \mathcal{R} is a set of rules $\ell \rightarrow r$ between terms $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. A TRS \mathcal{R} is linear if its rewrite rules consist of linear terms. We call \mathcal{R} *variable-separated* if $\text{Var}(\ell) \cap \text{Var}(r) = \emptyset$ for every $\ell \rightarrow r \in \mathcal{R}$.

In this paper we are concerned with finite, linear, variable-separated TRSs \mathcal{R} and we (mostly) consider rewriting on ground terms: $t \rightarrow_{\mathcal{R}} u$ for ground terms t, u if there exist a context C , a rewrite rule $\ell \rightarrow r \in \mathcal{R}$, and a substitution σ such that $t = C[\ell\sigma]$ and $u = C[r\sigma]$. We write $\rightarrow_{\mathcal{R}}^*$ for the reflexive and transitive closure of $\rightarrow_{\mathcal{R}}$. Further relations on terms will be introduced in the next section. We drop the subscript \mathcal{R} when it can be inferred from the context. A ground normal form is a ground term t such that $t \rightarrow_{\mathcal{R}} u$ for no term u . We write $\text{NF}(\mathcal{R})$ for the set of ground normal forms of \mathcal{R} .

Example 1 We use the TRS \mathcal{R} consisting of the rewrite rules

$$a \rightarrow b \qquad f(a) \rightarrow b \qquad g(a, x) \rightarrow f(a)$$

over the signature $\mathcal{F} = \{a, b, f, g\}$ as leading example in this paper. We have

$$f(g(a, b)) \rightarrow_{\mathcal{R}} f(f(a)) \rightarrow_{\mathcal{R}} f(b)$$

with ground normal form $f(b)$.

2.2 Tree Automata

A (finite bottom-up) tree automaton $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ consists of a finite signature \mathcal{F} , a finite set Q of states, disjoint from \mathcal{F} , a subset $Q_f \subseteq Q$ of final states, and a set of transition rules Δ . Every transition rule has one of the following two shapes:

- $f(p_1, \dots, p_n) \rightarrow q$ with $f \in \mathcal{F}$ and $p_1, \dots, p_n, q \in Q$, or
- $p \rightarrow q$ with $p, q \in Q$.

Transition rules of the second shape are called ε -transitions. Transition rules can be viewed as rewrite rules between ground terms in $\mathcal{T}(\mathcal{F} \cup Q, \mathcal{V})$. The induced rewrite relation is denoted by \rightarrow_{Δ} or $\rightarrow_{\mathcal{A}}$. A ground term $t \in \mathcal{T}(\mathcal{F})$ is accepted by \mathcal{A} if $t \rightarrow_{\Delta}^* q$ for some $q \in Q_f$. The set of all accepted terms is denoted by $L(\mathcal{A})$ and a set L of ground terms is regular if $L = L(\mathcal{A})$ for some tree automaton \mathcal{A} . A tree automaton \mathcal{A} is deterministic if there are no ε -transitions and no two transition rules with the same left-hand side. We say that \mathcal{A} is completely defined if it contains a transition rule with left-hand side $f(p_1, \dots, p_n)$ for every n -ary function symbol f and every combination p_1, \dots, p_n of states. All regular sets are accepted by a completely defined, deterministic tree automaton. The class of regular sets is effectively closed under Boolean operations. Moreover, membership and emptiness are decidable.

For relations on ground terms two different types of automata are used. The first one is restricted to binary relations. 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} . Let s and t be ground terms in $\mathcal{T}(\mathcal{F})$. We say that the pair (s, t) is accepted by \mathcal{G} if $s \rightarrow_{\mathcal{A}}^* u \xrightarrow{\mathcal{B}}^* t$ for some term $u \in \mathcal{T}(\mathcal{F} \cup Q)$. Here Q is the combined set of states of \mathcal{A} and \mathcal{B} . The set of all such pairs is denoted by $L(\mathcal{G})$. Observe that $L(\mathcal{G})$ is a binary relation on $\mathcal{T}(\mathcal{F})$. A binary relation \bowtie on ground terms is a GTT relation if there exists a GTT \mathcal{G} such that $\bowtie = L(\mathcal{G})$. In FORT we deal with *anchored* GTTs, which are GTTs with a different acceptance condition: A pair (s, t) of ground terms is accepted by an anchored GTT \mathcal{G} if $s \rightarrow_{\mathcal{A}}^* q \xrightarrow{\mathcal{B}}^* t$ for some (common) state q . The set of all such pairs is denoted by $L_a(\mathcal{G})$. It can be shown that the resulting language class coincides with binary Rec_{\times} which is defined in [8, Sect. 3.2.1] as the class of finite unions of Cartesian products of regular sets. The more operational view above benefits the developments described in subsequent sections. We obviously have $L_a(\mathcal{G}) \subseteq L(\mathcal{G})$. Anchored GTT relations have the advantage that they can represent the root-step relation $\rightarrow_{\varepsilon}$, which is not possible with GTT relations as the latter are always reflexive. Moreover, they have better closure properties than GTT relations. When we speak of “anchored GTTs”, we always have $L_a(\mathcal{G})$ in mind.

The second method for representing relations on ground terms uses standard tree automata operating on an encoding of the relation as a set of ground terms over a special signature. For a signature \mathcal{F} and $n \geq 0$ we let $\mathcal{F}^{(n)} = (\mathcal{F} \cup \{\perp\})^n$. Here, $\perp \notin \mathcal{F}$ is a fresh constant. The arity of a symbol $f_1 \dots f_n \in \mathcal{F}^{(n)}$ is the maximum of the arities of f_1, \dots, f_n and 0 if $n = 0$. Given n terms $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})$, the term $\langle t_1, \dots, t_n \rangle$ is the unique term $u \in \mathcal{T}(\mathcal{F}^{(n)})$ such

that $\mathcal{Pos}(u) = \mathcal{Pos}(t_1) \cup \dots \cup \mathcal{Pos}(t_n)$ and $u(p) = f_1 \dots f_n$ where $f_i = t_i(p)$ if $p \in \mathcal{Pos}(t_i)$ and \perp otherwise, for all positions $p \in \mathcal{Pos}(u)$. If $n = 0$ then $\mathcal{Pos}(u) = \{\varepsilon\}$ and $u(\varepsilon)$ is the empty sequence.

Example 2 For $\mathcal{F} = \{a, b, f, g\}$ in Example 1 we have

$$\begin{aligned} \langle g(a, f(b)), f(a) \rangle &= gf(aa, f\perp(b\perp)) \in \mathcal{T}(\mathcal{F}^{(2)}) \\ \langle a, f(f(b)), g(b, a) \rangle &= afg(\perp fb(\perp b\perp), \perp\perp a) \in \mathcal{T}(\mathcal{F}^{(3)}) \end{aligned}$$

An n -ary relation R on $\mathcal{T}(\mathcal{F})$ is regular if its encoding $\{\langle t_1, \dots, t_n \rangle \mid (t_1, \dots, t_n) \in R\}$ is regular. The class of all n -ary regular relations is denoted by RR_n . Every (anchored) GTT relation belongs to RR_2 . The well-known construction (presented later in the proof of Theorem 10) is used to decide membership for anchored GTT relations.

3 First-Order Theory of Rewriting

We consider first-order logic over a language \mathcal{L} without function symbols. The language contains the following binary predicate symbols:

$$\rightarrow \qquad \rightarrow^* \qquad =$$

Further predicate symbols will be added to \mathcal{L} later in this paper. As models we consider finite linear variable-separated 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 \rightarrow in $(\mathcal{F}, \mathcal{R})$ is the one-step rewrite relation $\rightarrow_{\mathcal{R}}$ over $\mathcal{T}(\mathcal{F})$, \rightarrow^* denotes its transitive-reflexive closure, and $=$ is interpreted as equality on ground terms.

Variable-separated TRSs appear naturally when approximating TRSs that satisfy the usual variable restriction $(\mathcal{V}\text{ar}(r) \subseteq \mathcal{V}\text{ar}(\ell)$ for every rewrite rule $\ell \rightarrow r$), to achieve regularity of the set of reachable terms starting from a regular set of ground terms. The support for linear variable-separated TRSs opens up the possibility of using FORT to compute dependency graphs based on the non-variable approximation for termination analysis [40], check infeasibility of conditional critical pairs for confluence analysis of conditional TRSs [51], and compute needed redexes based on the strong and non-variable approximations for the analysis of optimal normalizing strategies [18].

The following example gives an idea of the decision procedure for the first-order theory of rewriting. It shows how (closure) operations on tree automata and GTTs are used to obtain tree automata, each of which represent tuples of ground terms satisfying subformulas of the formula of interest. These operations are presented in Sect. 5 together with correctness proofs that have been formalized.

Example 3 Consider the formula

$$\varphi = \forall s \exists t (s \rightarrow^* t \wedge \neg \exists u (t \rightarrow u))$$

which expresses the normalization property of TRSs. To determine whether a given linear variable-separated TRS \mathcal{R} over a signature \mathcal{F} satisfies φ , we construct automata for the subterms of the formula in a bottom-up fashion. We start with an RR_1 automaton \mathcal{A}_1 that

accepts the ground normal forms in $\mathcal{T}(\mathcal{F})$, using an algorithm first described in [6] and covered in Sect. 5.4:

$$\text{RR}_1 \ A_1 \quad L(A_1) = \{t \mid t \in \text{NF}(\mathcal{R})\} \quad (\text{Theorem 15})$$

Here $t \in \text{NF}(\mathcal{R})$ stands for $\neg \exists u (t \rightarrow u)$. Next we construct an anchored GTT \mathcal{G}_1 accepting the root-step relation of \mathcal{R} :

$$\text{GTT} \ G_1 \quad L_a(G_1) = \{(s, t) \mid s \rightarrow_\varepsilon t\} \quad (\text{Theorem 4})$$

Using a modified transitive closure operation, we obtain an anchored GTT \mathcal{G}_2 :

$$\text{GTT} \ G_2 \quad L_a(G_2) = \{(s, t) \mid s \rightarrow^* \cdot \rightarrow_\varepsilon \cdot \rightarrow^* t\} \quad (\text{Theorem 8})$$

Since anchored GTT relations are also RR_2 relations we can construct an equivalent RR_2 automaton A_2 :

$$\text{RR}_2 \ A_2 \quad L(A_2) = \{\langle s, t \rangle \mid s \rightarrow^* \cdot \rightarrow_\varepsilon \cdot \rightarrow^* t\} \quad (\text{Theorem 10})$$

Using a special context closure operation, we obtain an RR_2 automaton A_3 accepting the encoding of \rightarrow^* :

$$\text{RR}_2 \ A_3 \quad L(A_3) = \{\langle s, t \rangle \mid s \rightarrow^* t\} \quad (\text{Theorem 11})$$

Before the conjunction in $s \rightarrow^* t \wedge t \in \text{NF}(\mathcal{R})$ can be constructed, the arities of the RR_2 automaton A_3 and the RR_1 automaton A_1 have to match. With this goal A_1 is cylindrified (C_1) to construct the RR_2 automaton A_4 . Here care has to be taken that not only the arities match, but also that terms, taking the place of variables shared by both formulas, are at the same position i in the encoding $\langle t_1, \dots, t_i, \dots, t_n \rangle$ of both automata:

$$\text{RR}_2 \ A_4 \quad L(A_4) = \{\langle s, t \rangle \mid t \in \text{NF}(\mathcal{R})\} \quad (\text{Theorem 14})$$

After this, the intersection with A_3 results in the RR_2 automaton A_5 that models the conjunction:

$$\text{RR}_2 \ A_5 \quad L(A_5) = \{\langle s, t \rangle \mid s \rightarrow^* t \wedge t \in \text{NF}(\mathcal{R})\} \quad (\text{Theorem 12})$$

Applying the second projection (Π_2 , which removes the second component) produces the RR_1 automaton A_6 :

$$\text{RR}_1 \ A_6 \quad L(A_6) = \{s \mid \exists t (s \rightarrow^* t \wedge t \in \text{NF}(\mathcal{R}))\} \quad (\text{Theorem 14})$$

At this point φ holds if and only if $L(A_6) = \mathcal{T}(\mathcal{F})$. In FORT the \forall quantifier is transformed into the equivalent $\neg \exists \neg$. Hence complementation is used to obtain an RR_1 automaton \mathcal{A}_7

$$\text{RR}_1 \ A_7 \quad L(A_7) = \{s \mid \neg \exists t (s \rightarrow^* t \wedge t \in \text{NF}(\mathcal{R}))\} \quad (\text{Theorem 13})$$

and the existential quantifier is implemented using projection. This gives an RR_0 automaton \mathcal{A}_8 which either accepts the empty relation \emptyset or the singleton set $\{()\}$ consisting of the nullary tuple $()$. The outermost negation gives rise to another complementation step. The final RR_0 automaton \mathcal{A}_9 is tested for emptiness: $L(\mathcal{A}_9) = \emptyset$ if and only if the TRS \mathcal{R} does not satisfy φ .

$$\begin{aligned}
 A &::= \rightarrow_\varepsilon \text{ (Theorem 4)} \mid A^- \text{ (Lemma 10)} \mid A \cup A \text{ (Lemma 10)} \\
 &\mid A^+ \text{ (Theorem 6)} \mid A^\hat{+} \text{ (Theorem 8)} \mid A \circ A \text{ (Theorem 5)} \\
 &\mid A \hat{\circ} A \text{ (Theorem 7)} \mid A^c \text{ (Theorem 9)} \mid A \cap A \text{ (Lemma 17)} \\
 &\mid T \times T \text{ (Lemma 8)} \\
 R &::= A \text{ (Theorem 10)} \mid R_p^n \text{ (Theorem 11)} \mid R \cup R \text{ (Theorem 12)} \\
 &\mid R \cap R \text{ (Theorem 12)} \mid R^- \text{ (Corollary 2)} \mid R^c \text{ (Theorem 13)} \\
 &\mid =_T \text{ (Lemma 18)} \\
 n &::= \geq \mid 1 \mid > \quad p ::= \geq \mid \varepsilon \mid > \\
 T &::= \mathcal{T}(\mathcal{F}) \text{ (Lemma 1)} \mid \text{NF} \text{ (Theorem 15)} \mid \text{INF}_R \text{ (Theorem 3)} \\
 &\mid T \cup T \text{ (Theorem 1)} \mid T \cap T \text{ (Theorem 1)} \mid T^c \text{ (Theorem 1)} \\
 &\mid \pi_1(R) \text{ (Theorem 2)} \mid \pi_2(R) \text{ (Theorem 2)}
 \end{aligned}$$

Fig. 1 Automata operations for the predicates in the first-order theory of rewriting

In order to express termination in the first-order theory of rewriting, we extend \mathcal{L} with the binary predicate symbol \rightarrow^+ (which denotes the transitive closure of \rightarrow) and the unary predicate defined below (which goes back to a technical report by Dauchet and Tison [11]).

Definition 1 Let \bowtie be an arbitrary binary relation on $\mathcal{T}(\mathcal{F})$. We write INF_{\bowtie} for the set $\{t \in \mathcal{T}(\mathcal{F}) \mid t \bowtie u \text{ for infinitely many terms } u \in \mathcal{T}(\mathcal{F})\}$.

If we instantiate INF_{\bowtie} by taking $\bowtie = \rightarrow^*$, we obtain the predicate $\text{INF}_{\rightarrow^*}$ that is satisfied by ground terms that have infinitely many reducts. By forbidding cycles, we obtain the formula

$$\neg \exists t (\text{INF}_{\rightarrow^*}(t) \vee t \rightarrow^+ t)$$

that expresses termination of finite variable-separated TRSs.

The grammar in Fig. 1 lists the formalized (closure) operations for the *predicates* in the first-order theory of rewriting. Here A are anchored GTT relations, R are RR_2 relations, and T are regular sets of ground terms. Some of the operations will be introduced in subsequent sections. The TRS \mathcal{R} enters the picture in three places. First of all, \rightarrow_ε is the root-step relation of \mathcal{R} . Secondly, NF denotes the set of ground normal forms of \mathcal{R} . Finally, $\mathcal{T}(\mathcal{F})$ denotes the set of ground terms, which depends on the signature \mathcal{F} of \mathcal{R} .

Every atomic subformula (predicate) will be represented as an RR_1 or RR_2 relation. The logical structure of *formulas* in the first-order theory of rewriting is taken care of by additional closure operations on RR_i relations.

4 Context Operations

In the next section we describe formalized automata constructions to decide the first-order theory of rewriting. To save considerable formalization efforts, we introduce a few primitives that operate on binary relations that are accepted by various kinds of tree automata. These primitives are sufficient to generate all binary rewrite relations supported by FORT. For

defining the semantics of the primitives, we introduce some context operations on binary relations in this section.

Definition 2 Let \mathcal{F} be a signature. A *multi-hole context* is an element of $\mathcal{T}(\mathcal{F} \uplus \{\square\})$ where \square is a fresh constant symbol, called *hole*. If C is a multi-hole context with $n \geq 0$ holes and t_1, \dots, t_n are terms in $\mathcal{T}(\mathcal{F})$ then $C[t_1, \dots, t_n]$ denotes the term in $\mathcal{T}(\mathcal{F})$ obtained from C by replacing the holes from left to right with t_1, \dots, t_n . We write \mathcal{C} for the set of all multi-hole contexts. Given a binary relation \bowtie on ground terms in $\mathcal{T}(\mathcal{F})$ and a set of multi-hole contexts $\mathcal{D} \subseteq \mathcal{C}$, we write $\mathcal{D}(\bowtie)$ for the relation $\{(C[t_1, \dots, t_n], C[u_1, \dots, u_n]) \mid C \in \mathcal{D} \text{ has } n \text{ holes and } t_i \bowtie u_i \text{ for all } 1 \leq i \leq n\}$.

We consider two ways to restrict multi-hole contexts: restricting the number of holes and restricting the position of the holes.

- We denote the set of multi-hole contexts with exactly one hole by \mathcal{C}^1 . The set of multi-hole contexts with at least one hole is denoted by $\mathcal{C}^>$. Moreover \mathcal{C}^{\geq} simply denotes \mathcal{C} .
- We denote the set of multi-hole contexts with the property that every hole occurs below the root position by $\mathcal{C}_{>}$. This includes the set $\mathcal{T}(\mathcal{F})$ of ground terms (which are multi-hole contexts without holes). Similarly, \mathcal{C}_ε denotes the set of multi-hole contexts with the property that every hole occurs at the root position. So $\mathcal{C}_\varepsilon = \{\square\} \cup \mathcal{T}(\mathcal{F})$. Moreover, \mathcal{C}_{\geq} simply denotes \mathcal{C} .

By combining both types of restrictions, we obtain nine ways for defining new binary relations.

Definition 3 Let \bowtie be a binary relation on $\mathcal{T}(\mathcal{F})$. Given a number constraint $n \in \{\geq, 1, >\}$ and a position constraint $p \in \{\geq, \varepsilon, >\}$, we define the binary relation $\bowtie_{\varepsilon, n}^p$ on $\mathcal{T}(\mathcal{F})$ as $(\mathcal{C}^n \cap \mathcal{C}_p)(\bowtie)$.

Note that $\bowtie_{\varepsilon}^{\geq} = \bowtie^=$ and $\bowtie_{\varepsilon}^1 = \bowtie_{\varepsilon}^> = \bowtie$, for any \bowtie . Here $\bowtie^= = \bowtie \cup \{=\}$ denotes the reflexive closure of \bowtie .

Example 4 Recall the TRS \mathcal{R} from our leading example and consider the multi-hole contexts

$$C_1 = \square \quad C_2 = f(\square) \quad C_3 = g(\square, a) \quad C_4 = g(\square, \square) \quad C_5 = f(a)$$

We have $C_1, C_2, C_3 \in \mathcal{C}^1$, $C_1, C_2, C_3, C_4 \in \mathcal{C}^>$, $C_1, C_5 \in \mathcal{C}_\varepsilon$, and $C_2, C_3, C_4, C_5 \in \mathcal{C}_{>}$. Moreover, $(C_2[a], C_2[b]) \in (\rightarrow_{\mathcal{R}})_{>}^1$ and $(C_4[a, a], C_4[b, b]) \notin (\rightarrow_{\mathcal{R}})_{>}^1$.

Because $\mathcal{C}_{\geq} = \mathcal{C}^{\geq} = \mathcal{C}$, the relation $\bowtie_{\varepsilon, n}^{\geq}$ is the multi-hole context closure of \bowtie . Using the root-step relation \rightarrow_ε induced by a linear, variable-separated TRS \mathcal{R} as \bowtie , we obtain eight different relations for $(\rightarrow_\varepsilon)_{\varepsilon, n}^p$:

$$\begin{aligned} (\rightarrow_\varepsilon)_{\varepsilon}^{\geq} &= \multimap & (\rightarrow_\varepsilon)_{\varepsilon}^1 &= \rightarrow & (\rightarrow_\varepsilon)_{\varepsilon}^{\geq} &= \dot{\multimap} \\ (\rightarrow_\varepsilon)_{\varepsilon}^{\geq} &= \rightarrow_\varepsilon^= & (\rightarrow_\varepsilon)_{\varepsilon}^1 &= \rightarrow_\varepsilon & (\rightarrow_\varepsilon)_{\varepsilon}^{\geq} &= \rightarrow_\varepsilon \\ (\rightarrow_\varepsilon)_{\varepsilon}^{\geq} &= \multimap_{>\varepsilon} & (\rightarrow_\varepsilon)_{\varepsilon}^1 &= \rightarrow_{>\varepsilon} & (\rightarrow_\varepsilon)_{\varepsilon}^{\geq} &= \dot{\multimap}_{>\varepsilon} \end{aligned}$$

Here \multimap denotes a parallel step (which is the multi-hole context closure of \rightarrow), $\dot{\multimap}$ a non-empty parallel step, $\multimap_{>\varepsilon}$ a parallel step where only redexes below the root are contracted, and $\dot{\multimap}_{>\varepsilon}$ a non-empty parallel step where only redexes below the root are contracted.

Example 5 Consider the term pairs $\pi_1 = (g(a, a), g(b, b))$, $\pi_2 = (g(a, a), f(a))$, and $\pi_3 = (g(a, a), g(a, a))$. We have $\pi_1, \pi_2, \pi_3 \in \multimap$, $\pi_1, \pi_2 \in \dot{\multimap}$, $\pi_1 \in \dot{\multimap}_{>\varepsilon}$, and $\pi_3 \in \multimap_{>\varepsilon} \setminus \dot{\multimap}_{>\varepsilon}$.

5 Formalized Tree Automata Constructions

In this section we present constructions on tree automata and (anchored) GTTs that are required for the decision procedure. Most of the results are known [8]. We give explicit proofs, providing detailed constructions that form the basis of the implementation of the decision procedure in FORT as well as the formalization in Isabelle.

Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ be a tree automaton. A state $q \in Q$ is *reachable* if $t \rightarrow_{\Delta}^* q$ for some term $t \in \mathcal{T}(\mathcal{F})$. We say that q is *productive* if $C[q] \rightarrow_{\Delta}^* q_f$ for some ground context C and final state $q_f \in Q_f$. The automaton \mathcal{A} is *trim* if all states are both reachable and productive. Any tree automaton can be transformed into an equivalent trim automaton. This result has been formalized in IsaFoR by Felgenhauer and Thiemann [21]. The construction preserves determinism. The following results are well-known.

Lemma 1 ($T ::= \mathcal{T}(\mathcal{F})$) *The set of ground terms over a finite signature \mathcal{F} is regular.* ✔

Theorem 1 ($T ::= T \cup T \mid T \cap T \mid T^c$) *The class of regular sets is effectively closed under union, intersection, and complement.* ✔✔✔

Before we turn to the infinity predicate ($T ::= \text{INF}_R$), we present an important closure operation on regular relations. Other closure operations will be presented in Sect. 5.3.

Definition 4 Let R be an n -ary relation over $\mathcal{T}(\mathcal{F})$. If $n \geq 1$ and $1 \leq i \leq n$ then the i -th projection of R is the relation $\Pi_i(R) = \{(t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_n) \mid (t_1, \dots, t_n) \in R\}$.

Note that Π_1 removes the first component of an RR_n relation. So for a binary regular relation R , $\Pi_1(R)$ coincides with $\pi_2(R)$ in the grammar in Fig. 1.

Theorem 2 ($T ::= \pi_1(R) \mid \pi_2(R)$) *The class of regular relations is effectively closed under projection.* ✔✔

Proof (construction) Let $\mathcal{A} = (\mathcal{F}^{(n)}, Q, Q_f, \Delta)$ be a tree automaton that accepts $\langle R \rangle$. Assume $n \geq 1$ and let $1 \leq i \leq n$. We construct a tree automaton that accept $\langle \Pi_i(R) \rangle$. We assume that all states of \mathcal{A} are reachable and define $\mathcal{A}_{\Pi_i} = (\mathcal{F}^{(n-1)}, Q, Q_f, \Delta_{\Pi_i})$ where Δ_{Π_i} is obtained from Δ by replacing every transition rule of the form

$$f_1 \cdots f_{i-1} f_i f_{i+1} \cdots f_n(p_1, \dots, p_m) \rightarrow q$$

with

$$f_1 \cdots f_{i-1} f_{i+1} \cdots f_n(p_1, \dots, p_k) \rightarrow q$$

provided $n = 1$ or $f_1 \cdots f_{i-1} f_{i+1} \cdots f_n \neq \perp^{n-1}$ for $n > 1$. Here $k \leq m$ is the arity of $f_1 \cdots f_{i-1} f_{i+1} \cdots f_n$. Epsilon transitions in Δ are not affected. Note that for $n = 1$ this results in an automaton over the signature containing only a single constant $()$ (the nullary tuple). The proof that $L(\mathcal{A}_{\Pi_i}) = \langle \Pi_i(R) \rangle$ is given at the end of Sect. 5.3. □

Example 6 Consider the tree automaton $\mathcal{A} = (\mathcal{F}^{(2)}, \{0, \dots, 6\}, \{6\}, \Delta)$ with $\mathcal{F} = \{a, b, f, g\}$ and Δ consisting of the transition rules

aa → 0	bb → 0	gg(0) → 0	ff(0, 0) → 0
ab → 1	bb → 1	gb(2) → 1	fb(2, 2) → 1
a⊥ → 2	b⊥ → 2	g⊥(2) → 2	f⊥(2, 2) → 2

$a\perp \rightarrow 3$	$\perp b \rightarrow 5$	$fg(1, 3) \rightarrow 6$	$gf(4, 5) \rightarrow 6$
$aa \rightarrow 4$	$gg(6) \rightarrow 6$	$ff(6, 0) \rightarrow 6$	$ff(0, 6) \rightarrow 6$

This automaton accepts the encoding of $\rightarrow_{\mathcal{R}}$ on $\mathcal{T}(\mathcal{F})$ induced by the TRS \mathcal{R} consisting of the rewrite rules

$$f(x, a) \rightarrow g(b) \qquad g(a) \rightarrow f(a, b)$$

For the first projection we obtain the automaton $\Pi_1(\mathcal{A})$ consisting of the transition rules

$a \rightarrow 0$	$b \rightarrow 0$	$g(0) \rightarrow 0$	$f(0, 0) \rightarrow 0$
$b \rightarrow 1$	$b \rightarrow 5$	$g(1) \rightarrow 6$	$f(4, 5) \rightarrow 6$
$a \rightarrow 4$	$g(6) \rightarrow 6$	$f(6, 0) \rightarrow 6$	$f(0, 6) \rightarrow 6$

Note that the third row of transitions in Δ disappeared completely. The rule $fg(1, 3) \rightarrow 6$ is transformed into $g(1) \rightarrow 6$, so state 3 is dropped. The second projection results in the automaton $\Pi_2(\mathcal{A})$ that accepts the reducible ground terms of \mathcal{R} :

$a \rightarrow 0$	$b \rightarrow 0$	$g(0) \rightarrow 0$	$f(0, 0) \rightarrow 0$
$a \rightarrow 1$	$b \rightarrow 1$	$g(2) \rightarrow 1$	$f(2, 2) \rightarrow 1$
$a \rightarrow 2$	$b \rightarrow 2$	$g(2) \rightarrow 2$	$f(2, 2) \rightarrow 2$
$a \rightarrow 3$		$f(1, 3) \rightarrow 6$	$g(4) \rightarrow 6$
$a \rightarrow 4$	$g(6) \rightarrow 6$	$f(6, 0) \rightarrow 6$	$f(0, 6) \rightarrow 6$

We now present a formalized proof of a version of the *pumping lemma* that we need for the infinity predicate INF_R (in the proof of Theorem 3 below).

Lemma 2 *Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ be a tree automaton and $t \xrightarrow{*}_{\Delta} q$ with $t \in \mathcal{T}(\mathcal{F})$ and $q \in Q$. If $\text{height}(t) > |Q|$ then there exist contexts C_1 and $C_2 \neq \square$, a term u , and a state p such that $t = C_1[C_2[u]]$, $u \xrightarrow{*}_{\Delta} p$, $C_2[p] \xrightarrow{*}_{\Delta} p$, and $C_1[p] \xrightarrow{*}_{\Delta} q$. ✔*

Proof From the assumptions $t \xrightarrow{*}_{\Delta} q$ and $\text{height}(t) > |Q|$ we obtain a sequence

$$(t_1, \dots, t_{n+1}, q_1, \dots, q_{n+1}, D_1, \dots, D_n)$$

consisting of ground terms, states, and non-empty contexts with $n > |Q|$ such that

- $t_i \xrightarrow{*}_{\Delta} q_i$ for all $i \leq n + 1$,
- $D_i[t_i] = t_{i+1}$ and $D_i[q_i] \xrightarrow{*}_{\Delta} q_{i+1}$ for all $i \leq n$, and
- $q_{n+1} = q$ and $t_{n+1} = t$

by a straightforward induction proof on t . Because $n > |Q|$ there exist indices $1 \leq i < j \leq n$ such that $q_i = q_j$. We construct the contexts $C_1 = D_n[\dots[D_j]\dots]$ and $C_2 = D_{j-1}[\dots[D_i]\dots]$. Note that $C_2 \neq \square$ as $i < j$. We obtain $C_2[q_i] \xrightarrow{*}_{\Delta} q_j$ and $C_1[q_j] \xrightarrow{*}_{\Delta} q_{n+1}$ by induction on the difference $j - i$. By letting $p = q_i = q_j$ and $u = t_i$ we obtain the desired result. □

5.1 Infinity Predicate

Below we show that INF_R is regular for every RR_2 relation R . The following definition originates from [11] and plays an important role in the proof.

Definition 5 Given a tree automaton $\mathcal{A} = (\mathcal{F}^{(2)}, Q, Q_f, \Delta)$, the set $Q_\infty \subseteq Q$ consists of all states $q \in Q$ such that $\langle \perp, t \rangle \rightarrow_\Delta^* q$ for infinitely many terms $t \in \mathcal{T}(\mathcal{F})$.

Example 7 Consider the binary relation

$$R = \{(f(a, g^n(b)), g^m(f(a, b))) \mid n = 2 \text{ and } m \geq 1 \text{ or } n \geq 3 \text{ and } m = 1\}$$

over $\mathcal{T}(\mathcal{F})$ with $\mathcal{F} = \{a, b, f, g\}$. Its encoding $\langle R \rangle$ is accepted by the automaton $\mathcal{A} = (\mathcal{F}^{(2)}, Q, Q_f, \Delta)$ with $Q = \{0, \dots, 11\}$, $Q_f = \{0\}$, and Δ consisting of the following transition rules:

$$\begin{array}{llll} fg(1, 2) \rightarrow 0 & \perp f(3, 4) \rightarrow 5 & g\perp(6) \rightarrow 2 & b\perp \rightarrow 7 \\ fg(8, 9) \rightarrow 0 & \perp g(5) \rightarrow 5 & g\perp(7) \rightarrow 6 & b\perp \rightarrow 11 \\ af(3, 4) \rightarrow 1 & \perp a \rightarrow 3 & g\perp(10) \rightarrow 9 & ag(5) \rightarrow 1 \\ af(3, 4) \rightarrow 8 & \perp b \rightarrow 4 & g\perp(11) \rightarrow 10 & g\perp(11) \rightarrow 11 \end{array}$$

For instance,

$$\begin{aligned} \langle f(a, g(g(b))), g(f(a, b)) \rangle &= fg(af(\perp a, \perp b), g\perp(g\perp(b\perp))) \\ &\rightarrow_\Delta^* fg(af(3, 4), g\perp(g\perp(7))) \rightarrow_\Delta^* fg(1, g\perp(6)) \rightarrow_\Delta fg(1, 2) \rightarrow_\Delta 0 \end{aligned}$$

but $\langle f(a, g(b)), f(a, b) \rangle = ff(aa, gb(b\perp))$ is not accepted. We have $Q_\infty = \{5\}$. State 5 is reached by $\langle \perp, g^n(f(a, b)) \rangle$ for all $n \geq 0$.

Definition 6 Given $\mathcal{A} = (\mathcal{F}^{(2)}, Q, Q_f, \Delta)$, we define the tree automaton

$$\mathcal{A}_\infty = (\mathcal{F}^{(2)}, Q \cup \bar{Q}, \bar{Q}_f, \Delta \cup \bar{\Delta})$$

Here \bar{Q} is a copy of Q where every state is dashed: $\bar{q} \in \bar{Q}$ if and only if $q \in Q$. For every transition rule $fg(q_1, \dots, q_n) \rightarrow q \in \Delta$ we have the following transition rules in $\bar{\Delta}$:

$$\begin{aligned} fg(q_1, \dots, q_n) &\rightarrow \bar{q} && \text{if } q \in Q_\infty \text{ and } f = \perp && (1) \\ fg(q_1, \dots, q_{i-1}, \bar{q}_i, q_{i+1}, \dots, q_n) &\rightarrow \bar{q} && \text{for all } 1 \leq i \leq n && (2) \end{aligned}$$

Moreover, for every ε -transition $p \rightarrow q \in \Delta$ we add

$$\bar{p} \rightarrow \bar{q} \tag{3}$$

to $\bar{\Delta}$. We write Δ' for $\Delta \cup \bar{\Delta}$.

Dashed states are created by rules of shape (1) and propagated by rules of shapes (2) and (3). The above construction differs from the one in [11]; instead of (1) the latter contains $fg(q_1, \dots, q_n) \rightarrow \bar{q}$ if $q_i \in Q_\infty$ for some $i > \text{arity}(f)$. In an implementation, rather than adding all dashed states and all transition rules of shape (2), the necessary rules would be computed by propagating the dashes created by (1) in order to avoid the appearance of unreachable dashed states. When \mathcal{A}_∞ is used in isolation, a single bit suffices to record that a dashed state occurred during a computation.

Example 8 For the tree automaton \mathcal{A} from Example 7 we obtain \mathcal{A}_∞ by adding the following transition rules (the missing rules of shape (2) involve unreachable states):

$$\perp f(3, 4) \rightarrow \bar{5} \quad \perp g(5) \rightarrow \bar{5} \quad \perp g(\bar{5}) \rightarrow \bar{5} \quad ag(\bar{5}) \rightarrow \bar{1} \quad fg(\bar{1}, 2) \rightarrow \bar{0}$$

The unique final state of \mathcal{A}_∞ is $\bar{0}$. We have $\langle f(a, g(g(b))), g(f(a, b)) \rangle \in L(\mathcal{A}_\infty)$ but there is no term u such that $\langle f(a(g(b))), u \rangle \in L(\mathcal{A}_\infty)$.

The following preliminary lemma is used in the proof of the theorem below and provides a characterization of the ground terms that reduce to a dashed state.

Lemma 3 *Let t be a term in $\mathcal{T}(\mathcal{F}^{(2)})$. If $t \rightarrow_{\mathcal{A}_\infty}^* \bar{p}$ then there exist a state $q \in Q_\infty$, a context C , and a term s such that $t = C[s]$, $\text{root}(s) = \perp f$ with $f \in \mathcal{F}$, $s \rightarrow_{\mathcal{A}_\infty}^* \bar{q}$, and $C[\bar{q}] \rightarrow_{\mathcal{A}_\infty}^* \bar{p}$. ✔*

Proof Write $t = gf(t_1, \dots, t_n)$. We distinguish two cases, depending on when the dash is introduced in $t \rightarrow_{\mathcal{A}_\infty}^* \bar{p}$. In the first case the dash is created by a root step:

$$t \rightarrow_{\Delta}^* gf(q_1, \dots, q_n) \rightarrow_{\Delta'} \bar{q} \rightarrow_{\Delta'}^* \bar{p}$$

We have $g = \perp$ and $q \in Q_\infty$ by (1). Hence we can take $s = t$ and $C = \square$. Note that $\text{root}(s) = gf = \perp f$. In the second case the dash is created during the evaluation of an argument t_i of t , and hence the given sequence $t \rightarrow_{\mathcal{A}_\infty}^* \bar{p}$ can be rearranged as

$$t \rightarrow_{\mathcal{A}_\infty}^* gf(t_1, \dots, \bar{r}, \dots, t_n) \rightarrow_{\mathcal{A}_\infty}^* \bar{p}$$

The induction hypothesis yields a state $q \in Q_\infty$, a context C' , and a term s such that $t_i = C'[s]$, $\text{root}(s) = \perp f'$ with $f' \in \mathcal{F}$, $s \rightarrow_{\mathcal{A}_\infty}^* \bar{q}$, and $C'[\bar{q}] \rightarrow_{\mathcal{A}_\infty}^* \bar{r}$. In this case we simply take $C = t[C']_i = gf(t_1, \dots, C', \dots, t_n)$. We have $t = t[t_i]_i = t[C'[s]]_i = C[s]$ and $C[\bar{q}] = gf(t_1, \dots, C'[\bar{q}], \dots, t_n) \rightarrow_{\mathcal{A}_\infty}^* gf(t_1, \dots, \bar{r}, \dots, t_n) \rightarrow_{\mathcal{A}_\infty}^* \bar{p}$. □

The following result goes back to a technical report by Dauchet and Tison [11].

Theorem 3 ($T ::= \text{INF}_R$) *The set INF_R is regular for every RR_2 relation R . ✔✔*

Proof Let $\mathcal{A} = (\mathcal{F}^{(2)}, Q, Q_f, \Delta)$ be a tree automaton that accepts $\langle R \rangle$. We show that $\text{INF}_R = \Pi_2(L(\mathcal{A}_\infty))$. The regularity of INF_R then follows from Theorem 2.

First suppose $t \in \text{INF}_R$. So $\langle t, u \rangle \in L(\mathcal{A})$ for infinitely many terms $u \in \mathcal{T}(\mathcal{F})$. Since the signature \mathcal{F} is finite, there are only finitely many ground terms of any given height. Moreover, $\text{height}(\langle t, u \rangle) = \max(\text{height}(t), \text{height}(u))$. Hence there must exist a term $u \in \mathcal{T}(\mathcal{F})$ with $\langle t, u \rangle \in L(\mathcal{A})$ such that $\text{height}(t) + |Q| + 1 < \text{height}(u)$. This is only possible if there are positions p and q such that $p \notin \text{Pos}(t)$, $pq \in \text{Pos}(u)$, and $|Q| < |q|$. From $\text{Pos}(\langle t, u \rangle) = \text{Pos}(t) \cup \text{Pos}(u)$ we obtain $\langle t, u \rangle|_p = \langle \perp, u|_p \rangle$. Since $\langle t, u \rangle \in L(\mathcal{A})$ there exist states $r \in Q$ and $q_f \in Q_f$ such that

$$\langle t, u \rangle = \langle t, u \rangle[\langle \perp, u|_p \rangle]_p \rightarrow_{\mathcal{A}}^* \langle t, u \rangle[r]_p \rightarrow_{\mathcal{A}}^* q_f$$

where we assume without loss of generality that the final step in the subsequence $\langle \perp, u|_p \rangle \rightarrow_{\mathcal{A}}^* r$ uses a non- ε -transition rule. From $|Q| < |q|$ and $pq \in \text{Pos}(u)$ we infer $|Q| < \text{height}(\langle \perp, u|_p \rangle)$. Hence we can use the pumping lemma (Lemma 2) to conclude the existence of infinitely many terms $v \in \mathcal{T}(\mathcal{F})$ such that $\langle \perp, v \rangle \rightarrow_{\mathcal{A}}^* r$. Hence $r \in Q_\infty$ by Definition 5. Since the final step in $\langle \perp, u|_p \rangle \rightarrow_{\mathcal{A}}^* r$ uses a non- ε -transition rule, we obtain $\langle \perp, u|_p \rangle \rightarrow_{\mathcal{A}_\infty}^* \bar{r}$ from the construction of \mathcal{A}_∞ with a final application of a rule of shape (1). We obtain $\langle t, u \rangle[\bar{r}]_p \rightarrow_{\mathcal{A}_\infty}^* \bar{q}_f$ from $\langle t, u \rangle[r]_p \rightarrow_{\mathcal{A}}^* q_f$. Hence $\langle t, u \rangle \rightarrow_{\mathcal{A}_\infty}^* \bar{q}_f$ and since $\bar{q}_f \in \bar{Q}_f$, $\langle t, u \rangle \in L(\mathcal{A}_\infty)$ and thus $t \in \Pi_2(L(\mathcal{A}_\infty))$.

Next suppose $t \in \Pi_2(L(\mathcal{A}_\infty))$. So $\langle t, u \rangle \in L(\mathcal{A}_\infty)$ for some ground terms u . There exists a final state $\bar{q}_f \in \bar{Q}$ with $\langle t, u \rangle \rightarrow_{\mathcal{A}_\infty}^* \bar{q}_f$. Using Lemma 3, we obtain a context C , a term s with $\text{root}(s) = \perp f$ for some $f \in \mathcal{F}$, and a state $q \in Q_\infty$ such that $C[s] = \langle t, u \rangle$, $s \rightarrow_{\mathcal{A}_\infty}^* \bar{q}$, and $C[\bar{q}] \rightarrow_{\mathcal{A}_\infty}^* \bar{q}_f$. Let p be the position of the hole in C . From $C[s] = \langle t, u \rangle$ and $\text{root}(s) = \perp f$, we infer $p \in \text{Pos}(u) \setminus \text{Pos}(t)$. Since $q \in Q_\infty$ the set $\{v \in \mathcal{T}(\mathcal{F}) \mid \langle \perp, v \rangle \rightarrow_{\mathcal{A}}^* q\}$ is

$$\frac{\perp f(p_1, \dots, p_n) \rightarrow_{\Delta} p}{p_1 \rightsquigarrow p \quad \dots \quad p_n \rightsquigarrow p} \qquad \frac{p \rightsquigarrow q \quad q \rightarrow_{\Delta} r}{p \rightsquigarrow r} \qquad \frac{p \rightsquigarrow q \quad q \rightsquigarrow r}{p \rightsquigarrow r}$$

Fig. 2 Inference rules for computing Q_{∞}^e

infinite. Hence the set $S = \{u[v]_p \in \mathcal{T}(\mathcal{F}) \mid \langle \perp, v \rangle \rightarrow_{\mathcal{A}}^* q\}$ is infinite, too. Let $u[w]_p \in S$. So $\langle \perp, w \rangle \rightarrow_{\mathcal{A}}^* q$. We obtain $C[q] \rightarrow_{\mathcal{A}}^* q_f$ from $C[\bar{q}] \rightarrow_{\mathcal{A}_{\infty}}^* \bar{q}_f$ by erasing all dashes. We have $C[w] = \langle t, u[w]_p \rangle$ as $p \in \text{Pos}(u) \setminus \text{Pos}(t)$. It follows that $\langle t, u[w]_p \rangle \in L(\mathcal{A})$ and thus there are infinitely many terms u' such that $\langle t, u' \rangle \in L(\mathcal{A})$. Since $\langle R \rangle = L(\mathcal{A})$ we conclude $t \in \text{INF}_R$ as desired. \square

Due to the definition of Q_{∞} , the automaton \mathcal{A}_{∞} defined in Definition 6 is not executable. We present an equivalent but executable definition, which we name Q_{∞}^e :

$$Q_{\infty}^e = \{q \mid p \rightsquigarrow p \text{ and } p \rightsquigarrow q \text{ for some state } p \in Q\}$$

Here the relation \rightsquigarrow is defined using the inference rules in Fig. 2. Intuitively, the first rule initializes the relation. Finding a cycle $p \rightsquigarrow^+ p$ ensures the existence of infinitely many terms $\langle \perp, s \rangle$ that reduce to p . The other two rules are used to collapse cycles (and other non-empty sequences of ε -transitions) into single steps.

Before proving that the two definitions are equivalent, we illustrate the definition of Q_{∞}^e by revisiting Example 7.

Example 9 We obtain $3 \rightsquigarrow 5$ and $4 \rightsquigarrow 5$ by applying the first inference rule to the transition rule $\perp f(3, 4) \rightarrow 5$. Similarly, $\perp g(5) \rightarrow 5$ gives rise to $5 \rightsquigarrow 5$. Since \mathcal{A} has no ε -transitions, no further inferences can be made. It follows that $Q_{\infty}^e = \{5\}$.

We call a term in $\mathcal{T}(\{\perp\} \times \mathcal{F})$ *right-only*. A term in $\mathcal{T}(\{\perp\} \times \mathcal{F}) \cup \{\square\}$ with exactly one occurrence of the hole \square is a *right-only context*.

Definition 7 We denote the composition of $\rightarrow_{\Delta-\varepsilon}$ and $\rightarrow_{\Delta\varepsilon}^*$ by \rightarrow_{Δ} .

The proof of the next lemma is straightforward. Note that the relations \rightarrow_{Δ}^* and \rightarrow_{Δ}^* do not coincide on *mixed* terms, involving function symbols and states.

Lemma 4 Let C be a ground context. We have $C[p] \rightarrow_{\Delta}^* q$ if and only if $p \rightarrow_{\Delta}^* p'$ and $C[p'] \rightarrow_{\Delta}^* q$ for some state p' . ✓

Proof First we show $t \rightarrow_{\Delta}^* q$ if $t \rightarrow_{\Delta}^* q$, for all ground terms t and states q . We use induction on $t = f(t_1, \dots, t_n)$. The given derivation $t \rightarrow_{\Delta}^* q$ may be written as $t \rightarrow_{\Delta}^* f(q_1, \dots, q_n) \rightarrow_{\Delta-\varepsilon} q' \rightarrow_{\Delta}^* q$. We obtain $t_i \rightarrow_{\Delta}^* q_i$ for $1 \leq i \leq n$ from the induction hypothesis. Clearly, $f(q_1, \dots, q_n) \rightarrow_{\Delta} q$ and hence $t \rightarrow_{\Delta}^* q$ as desired.

Next we prove the statement of the lemma. The if direction is trivial. For the only-if direction we use induction on the ground context C . Let $C[p] \rightarrow_{\Delta}^* q$. If $C = \square$ then we take $p' = q$. Suppose $C = f(t_1, \dots, C', \dots, t_n)$. We may write the derivation $C[p] \rightarrow_{\Delta}^* q$ as $t \rightarrow_{\Delta}^* f(q_1, \dots, q_n) \rightarrow_{\Delta-\varepsilon} q' \rightarrow_{\Delta}^* q$. The induction hypothesis yields a state p' such that $p \rightarrow_{\Delta}^* p'$ and $C'[p'] \rightarrow_{\Delta}^* q_i$ and we obtain $t_j \rightarrow_{\Delta}^* q_j$ for $j \neq i$ from the first part of the proof. We have $f(q_1, \dots, q_n) \rightarrow_{\Delta} q$ and hence $C[p'] = f(t_1, \dots, C'[p'], \dots, t_n) \rightarrow_{\Delta}^* q$. \square

Lemma 5 $Q_{\infty} \subseteq Q_{\infty}^e$ ✓

Proof We start by proving the following claim:

$$\text{if } C[p] \rightarrow_{\Delta}^* q \text{ and } C \text{ is a non-empty right-only context then } p \rightsquigarrow q \tag{4}$$

We use induction on the structure of C . If $C = \square$ there is nothing to show. Suppose $C = \perp f(t_1, \dots, C', \dots, t_n)$ where C' is the i -th subterm of C . The sequence $C[p] \rightarrow_{\Delta}^* q$ can be rearranged as $C[p] = \perp f(t_1, \dots, C'[p], \dots, t_n) \rightarrow_{\Delta}^* \perp f(q_1, \dots, q_n) \rightarrow_{\Delta} q' \rightarrow_{\Delta}^* q$. We obtain $q_i \rightsquigarrow q'$ and subsequently $q_i \rightsquigarrow q$ by using the inference rules in Fig. 2. If $C' = \square$ then $p = q_i$ and if $C' \neq \square$ then the induction hypothesis yields $p \rightsquigarrow q_i$ and thus $p \rightsquigarrow q$ by transitivity. This concludes the proof of (4).

Assume $q \in Q_{\infty}$, so there exist infinitely many terms t such that $\langle \perp, t \rangle \rightarrow_{\Delta}^* q$. Since the signature is finite, there exist terms of arbitrary height. Thus there exists an arbitrary but fixed term t such that the height of t is greater than the number of states of Q . Write $t = f(t_1, \dots, t_n)$. Since the height of t is greater than the number of the states in Q , there exist a subterm s of t , a state p , and contexts C_1 and $C_2 \neq \square$ such that

1. $\langle \perp, t \rangle = C_1[C_2[\langle \perp, s \rangle]]$,
2. $\langle \perp, s \rangle \rightarrow_{\Delta}^* p$,
3. $C_2[p] \rightarrow_{\Delta}^* p$, and
4. $C_1[p] \rightarrow_{\Delta}^* q$.

From Lemma 4 we obtain a state q' such that $p \rightarrow_{\Delta}^* q'$ and $C_2[q'] \rightarrow_{\Delta}^* p$. Hence $q' \rightsquigarrow p$ by (4). We obtain $q' \rightsquigarrow q$ from $q' \rightsquigarrow p$ in connection with the inference rule for ε -transitions. We perform a case analysis of the context C_1 .

- If $C_1 = \square$ then $p \rightarrow_{\Delta}^* q$ and thus $q' \rightsquigarrow q$ follows from $q' \rightsquigarrow p$ in connection with the inference rule for ε -transitions. Hence $q \in Q_{\infty}^e$.
- If $C_1 \neq \square$ then Lemma 4 yields a state q'' such that $p' \rightarrow_{\Delta}^* q''$ and $C_1[q''] \rightarrow_{\Delta}^* q$. Hence $q'' \rightsquigarrow q$ by (4). We also have $C_2[q'] \rightarrow_{\Delta}^* q''$ and thus $q' \rightsquigarrow q''$ by (4). We obtain $q' \rightsquigarrow q$ from the transitivity rule. Hence also in this case we obtain $q \in Q_{\infty}^e$. □

For the following lemma, we need the fact that \mathcal{A} can be assumed to be trim, so every state is productive and reachable. We may do so because Theorem 3 talks about regular relations, and any automaton that accepts the same language as \mathcal{A} will witness the fact that the given relation R is regular.

Lemma 6 $Q_{\infty}^e \subseteq Q_{\infty}$, provided that \mathcal{A} is trim. ✔

Proof In connection with the fact that \mathcal{A} accepts $R \subseteq \mathcal{T}(\mathcal{F}) \times \mathcal{T}(\mathcal{F})$, trimness of \mathcal{A} entails that any run $t \rightarrow_{\Delta}^* q$ is embedded into an accepting run $C[t] \rightarrow_{\Delta}^* C[q] \rightarrow_{\Delta}^* q_f \in Q_f$. So $C[t] = \langle u, v \rangle$ for some $(u, v) \in R$, and hence t must be a well-formed term. Moreover, if $\text{root}(t) = \perp f$ for some $f \in \mathcal{F}$ then $t = \langle \perp, u \rangle$ for some term $u \in \mathcal{T}(\mathcal{F})$. We now show the converse of claim (4) in the proof of Lemma 5 for the relation \rightarrow_{Δ}^* :

$$\text{if } p \rightsquigarrow q \text{ then } C[p] \rightarrow_{\Delta}^* q \text{ for some ground right-only context } C \neq \square \tag{5}$$

We prove the claim by induction on the derivation of $p \rightsquigarrow q$. First suppose $p \rightsquigarrow q$ is derived from the transition rule $\perp f(p_1, \dots, p_i, \dots, p_n) \rightarrow q$ in Δ with $p_i = p$. Because all states are reachable by well-formed terms, there exist terms $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})$ such that $\langle \perp, t \rangle \rightarrow_{\Delta}^* p_i$ for all $1 \leq i \leq n$. Let $C_1 = \perp f(\langle \perp, t_1 \rangle, \dots, \square, \dots, \langle \perp, t_n \rangle)$ where the hole is the i -th argument. We have $C_1[p] \rightarrow_{\Delta}^* \perp f(p_1, \dots, p_i, \dots, p_n) \rightarrow_{\Delta} q$. Next suppose $p \rightsquigarrow q$ is derived from $p \rightsquigarrow q'$ and $q' \rightarrow_{\Delta} q$. The induction hypothesis yields a ground right-only context $C \neq \square$ such that $C[p] \rightarrow_{\Delta}^* q'$. Hence also $C[p] \rightarrow_{\Delta}^* q$. Finally, suppose

$p \rightsquigarrow q$ is derived from $p \rightsquigarrow r$ and $r \rightsquigarrow q$. The induction hypothesis yields non-empty ground right-only contexts C_1 and C_2 such that $C_1[p] \rightarrow_{\Delta}^* r$ and $C_2[r] \rightarrow_{\Delta}^* q$. Hence $C[p] \rightarrow_{\Delta}^* q$ for the context $C = C_2[C_1]$. This concludes the proof of (5).

Now let $q \in Q_{\infty}^e$. So there exists a state p such that $p \rightsquigarrow p$ and $p \rightsquigarrow q$. Using (5), we obtain non-empty ground right-only contexts C_1 and C_2 such that $C_1[p] \rightarrow_{\Delta}^* p$ and $C_2[p] \rightarrow_{\Delta}^* q$. Since all states are reachable, there exists a ground term $t \in \mathcal{T}(\mathcal{F}^{(2)})$ such that $t \rightarrow_{\Delta}^* p$. Hence $C_2[t] \rightarrow_{\Delta}^* q$ and, by the observation made at the beginning of the proof, $C_2[t]$ is a well-formed term. Since C_2 is right-only, it follows that $t = \langle \perp, u \rangle$ for some term $u \in \mathcal{T}(\mathcal{F})$. Now consider the infinitely many terms $t_n = C_2[C_1^n[t]]$ for $n \geq 0$. We have $t_n \rightarrow_{\Delta}^* q$ and t_n is right-only by construction. Hence $q \in Q_{\infty}$. □

Corollary 1 *If \mathcal{A} is trim then $Q_{\infty}^e = Q_{\infty}$.* □

5.2 Anchored GTT Relations

Next we turn our attention to formalized constructions on (anchored) GTTs. Many of the results and automata constructions in this subsection are known. In the formalization we also employ an equivalent but more flexible definition of anchored GTT.

Definition 8 A pair automaton is a triple $\mathcal{P} = (Q, \mathcal{A}, \mathcal{B})$ where \mathcal{A}, \mathcal{B} are tree automata and $Q \subseteq Q_{\mathcal{A}} \times Q_{\mathcal{B}}$. We define $L(\mathcal{P}) = \{(s, t) \mid s \rightarrow_{\mathcal{A}}^* p \text{ and } t \rightarrow_{\mathcal{B}}^* q \text{ with } (p, q) \in Q\}$.

Lemma 7 *Anchored GTTs and pair automata are equivalent.* ✓

Proof If $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ is a GTT then $L_a(\mathcal{G}) = L(\mathcal{P})$ for the pair automaton $\mathcal{P} = (Q, \mathcal{A}, \mathcal{B})$ with $Q = \{(p, p) \mid p \in Q_{\mathcal{A}} \cap Q_{\mathcal{B}}\}$. Conversely, given a pair automaton $\mathcal{P} = (Q, \mathcal{A}, \mathcal{B})$, we first rename the states of \mathcal{B} to obtain an equivalent tree automaton \mathcal{B}' such that \mathcal{A} and \mathcal{B}' do not share states. We add an ε -transition $p \rightarrow q'$ to \mathcal{A} for every $(p, q) \in Q$, resulting in the tree automaton \mathcal{A}' . Here q' is the (renamed) state in \mathcal{B}' that corresponds to state q in \mathcal{B} . The GTT $\mathcal{G} = (\mathcal{A}', \mathcal{B}')$ satisfies $L_a(\mathcal{G}) = L(\mathcal{P})$. □

The above lemma will be used in the sequel without mention.

Lemma 8 $(A ::= T \times U)$ *If T and U are regular sets of ground terms then $T \times U$ is an anchored GTT relation.*

Proof Let $A = (\mathcal{F}, Q_A, Q_{fA}, \Delta_A)$ and $B = (\mathcal{F}, Q_B, Q_{fB}, \Delta_B)$ be tree automata that accept T and U . The set $T \times U$ is accepted by the pair automaton $\mathcal{P} = (Q, \mathcal{A}, \mathcal{B})$ with $Q = Q_{fA} \times Q_{fB}$. □

There are several ways to associate a GTT $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ with a linear variable-separated TRS \mathcal{R} . The one in [9] uses for each rewrite rule $\ell \rightarrow r$ of \mathcal{R} a unique interface state i , common to \mathcal{A} and \mathcal{B} , and transition rules and states specific to \mathcal{A} (\mathcal{B}) that accept all ground instances of ℓ (r) in state i . No states are shared between different rewrite rules. The resulting GTT accepts \Rightarrow and $\rightarrow_{\varepsilon}$ when viewed as an anchored GTT. The second way to associate a GTT with a linear variable-separated TRS \mathcal{R} originates from Dauchet et al. [12]. The resulting GTT accepts a relation in between \Rightarrow and \rightarrow^* . The construction that we formalized can be seen as a pair automaton version of the construction in [9].

Theorem 4 $[A ::= \rightarrow_{\varepsilon}]$ *The relation $\rightarrow_{\varepsilon}$ is an anchored GTT relation for every linear variable-separated TRS \mathcal{R} .* ✓

Proof Let \mathcal{R} be a linear variable-separated TRS over a signature \mathcal{F} . We denote the set of left-hand (right-hand) sides of the rules in \mathcal{R} by $\text{lhs}(\mathcal{R})$ ($\text{rhs}(\mathcal{R})$). Given a set of terms T , we write $s \trianglelefteq T$ if s is a subterm of some term in T . Given a term s we write \hat{s} for the ground term obtained from s by replacing each variable with a designated (fresh) constant $*$. Let Q be the set of states $\langle \hat{t} \rangle$ for each $t \trianglelefteq \text{lhs}(\mathcal{R}) \cup \text{rhs}(\mathcal{R})$. The set Δ_{lhs} consists of the transitions

$$f(\langle \hat{t}_1 \rangle, \dots, \langle \hat{t}_n \rangle) \rightarrow \langle \widehat{f(t_1, \dots, t_n)} \rangle$$

for every $f(t_1, \dots, t_n) \trianglelefteq \text{lhs}(\mathcal{R})$ and, if some term in $\text{lhs}(\mathcal{R})$ contains a variable,

$$f(\langle * \rangle, \dots, \langle * \rangle) \rightarrow \langle * \rangle$$

for every $f \in \mathcal{F}$. The set Δ_{rhs} is defined similarly, using $\text{rhs}(\mathcal{R})$ instead of $\text{lhs}(\mathcal{R})$ for generating the rules. We now define $\mathcal{P}_{\mathcal{R}} = (Q, \Delta_{\text{lhs}}, \Delta_{\text{rhs}})$ with $Q = \{ \langle \hat{\ell} \rangle, \langle \hat{r} \rangle \mid \ell \rightarrow r \in \mathcal{R} \}$. It is easy to prove that $L_a(\mathcal{P}_{\mathcal{R}}) = \rightarrow_{\varepsilon}$. \square

The other binary relations associated with a TRS \mathcal{R} (like $\Rightarrow_{\mathcal{R}}$ and $\Leftarrow_{\mathcal{R}}^*$) will be obtained from the root-step relation $\rightarrow_{\varepsilon}$ by automata constructions that operate on anchored GTT relations and RR_2 relations.

Example 10 The pair automaton $\mathcal{P}_{\mathcal{R}} = (Q, \mathcal{A}, \mathcal{B})$ constructed in the above proof consists of the transition rules

$$\begin{array}{llll} \Delta_A: & a \rightarrow \langle * \rangle & b \rightarrow \langle * \rangle & f(\langle * \rangle) \rightarrow \langle * \rangle & g(\langle * \rangle, \langle * \rangle) \rightarrow \langle * \rangle \\ & a \rightarrow \langle a \rangle & & f(\langle a \rangle) \rightarrow \langle f(a) \rangle & g(\langle a \rangle, \langle * \rangle) \rightarrow \langle g(a, *) \rangle \\ \Delta_B: & a \rightarrow \langle a \rangle & b \rightarrow \langle b \rangle & f(\langle a \rangle) \rightarrow \langle f(a) \rangle & \\ Q: & (\langle a \rangle, \langle b \rangle) & (\langle f(a) \rangle, \langle b \rangle) & (\langle g(a, *) \rangle, \langle f(a) \rangle) & \end{array}$$

and accepts the root-step relation $\rightarrow_{\varepsilon}$ of our leading TRS \mathcal{R} . The state pairs in Q are presented as ε -transitions and perform the transfer from left-hand sides to right-hand sides of \mathcal{R} . For instance, $g(a, f(f(b))) \rightarrow_{\varepsilon} f(a)$ is witnessed by $g(a, f(f(b))) \xrightarrow{*}_{\mathcal{A}} \langle g(a, *) \rangle \rightarrow \langle f(a) \rangle \xleftarrow{*}_{\mathcal{B}} f(a)$. To shorten the notation in subsequent examples, we number the states as follows:

$$0 = \langle * \rangle \quad 1 = \langle a \rangle \quad 2 = \langle f(a) \rangle \quad 3 = \langle g(a, *) \rangle \quad 4 = \langle b \rangle$$

Hence the transition rules are presented as follows:

$$\begin{array}{llll} \Delta_A: & a \rightarrow 0 & b \rightarrow 0 & f(0) \rightarrow 0 & g(0, 0) \rightarrow 0 \\ & a \rightarrow 1 & & f(1) \rightarrow 2 & g(1, 0) \rightarrow 3 \\ \Delta_B: & a \rightarrow 1 & b \rightarrow 4 & f(1) \rightarrow 2 & \\ Q: & (1,4) & (2,4) & (3,2) & \end{array}$$

To turn $\mathcal{P}_{\mathcal{R}}$ into an equivalent anchored GTT $\mathcal{G}_{\mathcal{R}} = (\mathcal{A}', \mathcal{B}')$ we rename states 1 and 2 in \mathcal{B} into 5 and 6 and add the pairs in Q as ε -transitions to \mathcal{A} , after applying the renaming to their targets:

$$\begin{array}{llll} \Delta'_A: & a \rightarrow 0 & b \rightarrow 0 & f(0) \rightarrow 0 & g(0, 0) \rightarrow 0 \\ & a \rightarrow 1 & & f(1) \rightarrow 2 & g(1, 0) \rightarrow 3 \\ & 1 \rightarrow 4 & 2 \rightarrow 4 & 3 \rightarrow 6 & \\ \Delta_B: & a \rightarrow 5 & b \rightarrow 4 & f(5) \rightarrow 6 & \end{array}$$

Next we turn to composition and transitive closure.

$$\begin{array}{c}
 \frac{q \mathcal{A} \leftarrow p \quad p \rightsquigarrow r}{q \rightsquigarrow r} \text{ [a]} \qquad \frac{p \rightsquigarrow q \quad q \rightarrow_{\mathcal{B}} r}{p \rightsquigarrow r} \text{ [b]} \\
 \\
 \frac{p \mathcal{A} \leftarrow f(p_1, \dots, p_n) \quad p_1 \rightsquigarrow q_1 \cdots p_n \rightsquigarrow q_n \quad f(q_1, \dots, q_n) \rightarrow_{\mathcal{B}} q}{p \rightsquigarrow q} \text{ [c]}
 \end{array}$$

Fig. 3 $\Delta_\varepsilon(\mathcal{A}, \mathcal{B})$

Definition 9 Given tree automata \mathcal{A} and \mathcal{B} , $\Delta_\varepsilon(\mathcal{A}, \mathcal{B})$ is the set of ε -transitions \rightsquigarrow defined by the inference rules in Fig. 3.

The inference rule [c] appeared first in [17]. Since there are only finitely many ε -transitions between states in Q , $\Delta_\varepsilon(\mathcal{A}, \mathcal{B})$ can be effectively computed. The next result provides a useful equivalent characterization (which is presented as a definition in [8, 12]).

Example 11 For the (anchored) GTT $\mathcal{G}_{\mathcal{R}}$ of Example 10, which will be referred to as $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ in the following, the set $\Delta_\varepsilon(\mathcal{A}, \mathcal{B})$ consists of the following seven ε -transitions:

$$\begin{array}{llll}
 0 \rightsquigarrow 5 & \text{[c]} & (0 \mathcal{A} \leftarrow a \rightarrow_{\mathcal{B}} 5) & 0 \rightsquigarrow 6 & \text{[c]} & (0 \mathcal{A} \leftarrow f(0) \rightsquigarrow f(5) \rightarrow_{\mathcal{B}} 6) \\
 1 \rightsquigarrow 5 & \text{[c]} & (1 \mathcal{A} \leftarrow a \rightarrow_{\mathcal{B}} 5) & 2 \rightsquigarrow 6 & \text{[c]} & (2 \mathcal{A} \leftarrow f(1) \rightsquigarrow f(5) \rightarrow_{\mathcal{B}} 6) \\
 0 \rightsquigarrow 4 & \text{[c]} & (0 \mathcal{A} \leftarrow b \rightarrow_{\mathcal{B}} 4) & 4 \rightsquigarrow 5 & \text{[a]} & (4 \mathcal{A} \leftarrow 1 \rightsquigarrow 5) \\
 & & & 4 \rightsquigarrow 6 & \text{[a]} & (4 \mathcal{A} \leftarrow 2 \rightsquigarrow 6)
 \end{array}$$

Since \mathcal{B} does not contain ε -transitions, the inference rule [b] is not used here.

Lemma 9 If \mathcal{A} and \mathcal{B} are tree automata over a signature \mathcal{F} then

$$\Delta_\varepsilon(\mathcal{A}, \mathcal{B}) = \{p \rightsquigarrow q \mid p \mathcal{A}^* \leftarrow t \rightarrow_{\mathcal{B}}^* q \text{ for some ground term } t \in \mathcal{T}(\mathcal{F})\} \quad \checkmark$$

Proof First suppose there exists a ground term $t \in \mathcal{T}(\mathcal{F})$ with $p \mathcal{A}^* \leftarrow t \rightarrow_{\mathcal{B}}^* q$ for states p of \mathcal{A} and q of \mathcal{B} . We show $p \rightsquigarrow q$ by induction on $t = f(t_1, \dots, t_n)$. The sequence $t \rightarrow_{\mathcal{A}}^* p$ can be written as $t \rightarrow_{\mathcal{A}}^* f(p_1, \dots, p_n) \rightarrow_{\mathcal{A}} p' \rightarrow_{\mathcal{A}}^* p$ with states p_1, \dots, p_n, p' of \mathcal{A} . Similarly, $t \rightarrow_{\mathcal{B}}^* f(q_1, \dots, q_n) \rightarrow_{\mathcal{B}} q' \rightarrow_{\mathcal{B}}^* q$ with states q_1, \dots, q_n, q' of \mathcal{B} . We have $p_i \mathcal{A}^* \leftarrow t_i \rightarrow_{\mathcal{B}}^* q_i$ and thus $p_i \rightsquigarrow q_i$ by the induction hypothesis, for $1 \leq i \leq n$. Hence we obtain $p' \rightsquigarrow q'$ by [c]. Repeated applications of the inference rules [a] and [b] in connection with $p' \rightarrow_{\mathcal{A}}^* p$ and $q' \rightarrow_{\mathcal{B}}^* q$ yields $p \rightsquigarrow q$. Hence $p \rightsquigarrow q \in \Delta_\varepsilon(\mathcal{A}, \mathcal{B})$ as desired.

Next suppose $p \rightsquigarrow q \in \Delta_\varepsilon(\mathcal{A}, \mathcal{B})$. We show the existence of a ground term $t \in \mathcal{T}(\mathcal{F})$ such that $p \mathcal{A}^* \leftarrow t \rightarrow_{\mathcal{B}}^* q$ by induction on the derivation of $p \rightsquigarrow q$. In the base case [c] is used with $p \mathcal{A} \leftarrow a$ and $a \rightarrow_{\mathcal{B}} q$ for some constant a and hence we can take $t = a$. For the induction step we consider three cases, depending on which inference rule is used to derive $p \rightsquigarrow q$. First suppose [c] is used. So there exist transition rules $f(p_1, \dots, p_n) \rightarrow p$ in \mathcal{A} and $f(q_1, \dots, q_n) \rightarrow q$ in \mathcal{B} such that $p_i \rightsquigarrow q_i$ for $1 \leq i \leq n$. The induction hypothesis yields ground terms t_1, \dots, t_n such that $p_i \mathcal{A}^* \leftarrow t_i \rightarrow_{\mathcal{B}}^* q_i$ for $1 \leq i \leq n$. Hence $p \mathcal{A}^* \leftarrow t \rightarrow_{\mathcal{B}}^* q$ for $t = f(t_1, \dots, t_n)$. Next suppose [a] is applied to derive $p \rightsquigarrow q$. So there exists a state p' such that $p \mathcal{A} \leftarrow p' \rightsquigarrow q$. The induction hypothesis yields a ground term $t \in \mathcal{T}(\mathcal{F})$ such that $p' \mathcal{A}^* \leftarrow t \rightarrow_{\mathcal{B}}^* q$ and hence also $p \mathcal{A}^* \leftarrow t \rightarrow_{\mathcal{B}}^* q$. The reasoning for [b] is the same. \square

Theorem 5 ($A ::= A \circ A$) Anchored GTT relations are effectively closed under composition. \checkmark

$$\frac{(p, q) \in Q}{p \rightsquigarrow q} \qquad \frac{p \rightsquigarrow q \quad (q, q') \in \Delta_\varepsilon(\mathcal{B}, \mathcal{A}) \quad q' \rightsquigarrow r}{p \rightsquigarrow r}$$

Fig. 4 $\Delta_+(\mathcal{P})$ for $\mathcal{P} = (Q, \mathcal{A}, \mathcal{B})$


Proof Let $\mathcal{P}_1 = (Q_1, \mathcal{A}_1, \mathcal{B}_1)$ and $\mathcal{P}_2 = (Q_2, \mathcal{A}_2, \mathcal{B}_2)$ be pair automata (operating on terms over the same signature). We construct the pair automaton $\mathcal{P} = (Q, \mathcal{A}_1, \mathcal{B}_2)$ with

$$Q = Q_1 \circ \Delta_\varepsilon(\mathcal{B}_1, \mathcal{A}_2) \circ Q_2$$

We claim that $L(\mathcal{P}) = L(\mathcal{P}_1) \circ L(\mathcal{P}_2)$. First let $(s, t) \in L(\mathcal{P})$. We have $s \xrightarrow{*}_{\mathcal{A}_1} p$ and $t \xrightarrow{*}_{\mathcal{B}_2} q$ for some $(p, q) \in Q$. The definition of Q_2 yields states p' and q' such that $(p, p') \in Q_1$, $(p', q') \in \Delta_\varepsilon(\mathcal{B}_1, \mathcal{A}_2)$, and $(q', q) \in Q_2$. According to Lemma 9 there exists a ground term u such that $u \xrightarrow{*}_{\mathcal{B}_1} p'$ and $u \xrightarrow{*}_{\mathcal{A}_2} q'$. Hence $(s, u) \in L(\mathcal{P}_1)$ and $(u, t) \in L(\mathcal{P}_2)$ and thus $(s, t) \in L(\mathcal{P}_1) \circ L(\mathcal{P}_2)$.

For the converse, let $(s, t) \in L(\mathcal{P}_1) \circ L(\mathcal{P}_2)$. So there exists a ground term u such that $(s, u) \in L(\mathcal{P}_1)$ and $(u, t) \in L(\mathcal{P}_2)$. Hence there are pairs $(p_1, q_1) \in Q_1$ and $(p_2, q_2) \in Q_2$ such that $s \xrightarrow{*}_{\mathcal{A}_1} p_1$, $u \xrightarrow{*}_{\mathcal{B}_1} q_1$, $u \xrightarrow{*}_{\mathcal{A}_2} p_2$, and $t \xrightarrow{*}_{\mathcal{B}_2} q_2$. Lemma 9 yields $(q_1, p_2) \in \Delta_\varepsilon(\mathcal{B}_1, \mathcal{A}_2)$. Hence $(p_1, q_2) \in Q$ and thus $(s, t) \in L(\mathcal{P})$. \square

Example 12 We compose the pair automaton $\mathcal{P}_{\mathcal{R}} = (Q, \mathcal{A}, \mathcal{B})$ of Example 10 with itself. We have $\Delta_\varepsilon(\mathcal{B}, \mathcal{A}) = \Delta_\varepsilon(\mathcal{A}, \mathcal{B})^- = \{(1, 0), (1, 1), (4, 0), (2, 2), (2, 0)\}$. Hence we obtain the pair automaton $\mathcal{P}' = (Q', \mathcal{A}, \mathcal{B})$ with $Q' = Q \circ \Delta_\varepsilon(\mathcal{B}, \mathcal{A}) \circ Q = \{(3, 4)\}$. We have $L(\mathcal{A}, 3) = \{\mathbf{g}(\mathbf{a}, t) \mid t \in \mathcal{T}(\mathcal{F})\}$ and $L(\mathcal{B}, 4) = \{\mathbf{b}\}$. Hence, we obtain $L(\mathcal{P}') = L(\mathcal{A}, 3) \times L(\mathcal{B}, 4) = \rightarrow_{\varepsilon}^2$ as expected.

Theorem 6 ($A ::= A^+$) *Anchored GTT relations are effectively closed under transitive closure.* 

Proof Let $\mathcal{P} = (Q, \mathcal{A}, \mathcal{B})$ be a pair automaton. We construct the pair automaton $\mathcal{P}_+ = (\Delta_+(\mathcal{P}), \mathcal{A}, \mathcal{B})$ where $\Delta_+(\mathcal{P})$ is the binary relation on states defined by the inference rules in Fig. 4. We claim that $L(\mathcal{P}_+) = L(\mathcal{P})^+$. From the first inference rule we immediately obtain $L(\mathcal{P}) \subseteq L(\mathcal{P}_+)$. The second inference rule, together with the definition of Q in the proof of Theorem 5, yields $L(\mathcal{P}_+) \circ L(\mathcal{P}_+) \subseteq L(\mathcal{P}_+)$. Hence $L(\mathcal{P})^+ \subseteq L(\mathcal{P}_+)$.

For the converse, let $(s, t) \in L(\mathcal{P}_+)$. So there exists a pair $p \rightsquigarrow q$ such that $s \xrightarrow{*}_{\mathcal{A}} p$ and $t \xrightarrow{*}_{\mathcal{B}} q$. We prove $(s, t) \in L(\mathcal{P})^+$ by induction on the derivation of $p \rightsquigarrow q$. If $(p, q) \in Q$ then $(s, t) \in L(\mathcal{P})$. Suppose $p \rightsquigarrow p'$, $(p', q') \in \Delta_\varepsilon(\mathcal{B}, \mathcal{A})$, and $q' \rightsquigarrow q$. According to Lemma 9 there exists a ground term u such that $u \xrightarrow{*}_{\mathcal{B}} p'$ and $u \xrightarrow{*}_{\mathcal{A}} q'$. The induction hypothesis yields $(s, u) \in L(\mathcal{P})^+$ and $(u, t) \in L(\mathcal{P})^+$. Hence also $(s, t) \in L(\mathcal{P})^+$. \square

Example 13 Consider the pair automaton $\mathcal{P}_{\mathcal{R}} = (Q, \mathcal{A}, \mathcal{B})$ of Example 10. As observed in Example 12, $\Delta_\varepsilon(\mathcal{B}, \mathcal{A}) = \{(1, 0), (1, 1), (4, 0), (2, 2), (2, 0)\}$. Hence we obtain the pair automaton $\mathcal{P}_+ = (\Delta_+(\mathcal{P}), \mathcal{A}, \mathcal{B})$ with $\Delta_+(\mathcal{P}) = \{(1, 4), (2, 4), (3, 2), (3, 4)\}$. The pair $(3, 4)$ is obtained from the second inference rules with $p = 3, q = q' = 2$ and $r = 4$. We have $\mathbf{g}(\mathbf{a}, \mathbf{b}) \rightarrow_{\varepsilon} \mathbf{f}(\mathbf{a}) \rightarrow_{\varepsilon} \mathbf{b}$ and the pair $(\mathbf{g}(\mathbf{a}, \mathbf{b}), \mathbf{b})$ is accepted by \mathcal{P}_+ as $\mathbf{g}(\mathbf{a}, \mathbf{b}) \xrightarrow{*}_{\mathcal{A}} 3$ and $\mathbf{b} \rightarrow_{\mathcal{B}} 4$ with $(3, 4) \in \Delta_+(\mathcal{P})$. Furthermore, $\mathbf{g}(\mathbf{a}, \mathbf{b}) \rightarrow_{\varepsilon} \mathbf{f}(\mathbf{a}) \rightarrow \mathbf{f}(\mathbf{b})$ but $\mathbf{g}(\mathbf{a}, \mathbf{b}) \not\rightarrow_{\varepsilon}^+ \mathbf{f}(\mathbf{b})$ does not hold, and one readily checks that the pair $(\mathbf{g}(\mathbf{a}, \mathbf{b}), \mathbf{f}(\mathbf{b}))$ is not accepted by \mathcal{P}_+ .

Two further closure operations on anchored GTT relations are inverse and union. Recall that GTT relations are not closed under union.

Lemma 10 ($A ::= A^- \mid A \cup A$) *Anchored GTT relations are effectively closed under inverse and union.* ✓

Proof Given a pair automaton $\mathcal{P} = (Q, \mathcal{A}, \mathcal{B})$, we have $L(\mathcal{P})^- = L(\mathcal{P}^-)$ for the pair automaton $\mathcal{P}^- = (Q^-, \mathcal{B}, \mathcal{A})$. Here $Q^- = \{(q, p) \mid (p, q) \in Q\}$. Given pair automata $\mathcal{P}_1 = (Q_1, \mathcal{A}_1, \mathcal{B}_1)$ and $\mathcal{P}_2 = (Q_2, \mathcal{A}_2, \mathcal{B}_2)$ without common states, $L(\mathcal{P}_1) \cup L(\mathcal{P}_2) = L(\mathcal{P})$ for the pair automaton $\mathcal{P} = (Q_1 \cup Q_2, \mathcal{A}_1 \cup \mathcal{A}_2, \mathcal{B}_1 \cup \mathcal{B}_2)$. □

Next we present a modified composition operation $\hat{\circ}$ that preserves anchored GTT relations.

Definition 10 Given two binary relations \bowtie_1 and \bowtie_2 on the same set of ground terms, their modified composition $\bowtie_1 \hat{\circ} \bowtie_2$ is defined as the relation

$$\bowtie_1 \hat{\circ} \bowtie_2 = \bowtie_1 \circ (\bowtie_2)_{\geq}^{\geq} \cup (\bowtie_1)_{\geq}^{\geq} \circ \bowtie_2$$

We have $(\bowtie_1 \hat{\circ} \bowtie_2)_{\geq}^{\geq} = (\bowtie_1)_{\geq}^{\geq} \circ (\bowtie_2)_{\geq}^{\geq}$. The proof that anchored GTT relations are closed under $\hat{\circ}$ requires a preliminary result on the interplay between GTTs and anchored GTTs.

Lemma 11 *The composition of an anchored GTT relation and a GTT relation is an anchored GTT relation.*

Proof Let $\mathcal{P} = (Q, \mathcal{A}_1, \mathcal{B}_1)$ be a pair automaton and $\mathcal{G} = (\mathcal{A}_2, \mathcal{B}_2)$ a GTT. Without loss of generality we assume that \mathcal{P} and \mathcal{G} do not share states. Define the pair automaton

$$\mathcal{P}' = (Q, \mathcal{A}_1, \mathcal{B}_1 \cup \Delta_\varepsilon(\mathcal{A}_2, \mathcal{B}_1) \cup \mathcal{B}_2)$$

We claim that $L(\mathcal{P}') = L(\mathcal{P}) \circ L(\mathcal{G})$. First let $(s, t) \in L(\mathcal{P}')$. So $s \rightarrow_{\mathcal{A}_1}^* p$ and $t \rightarrow_{\mathcal{B}'}^* q$ with $(p, q) \in Q$ and \mathcal{B}' abbreviating $\mathcal{B}_1 \cup \Delta_\varepsilon(\mathcal{A}_2, \mathcal{B}_1) \cup \mathcal{B}_2$. Because \mathcal{P} and \mathcal{G} do not share states, the sequence $t \rightarrow_{\mathcal{B}'}^* q$ can be rearranged as follows:

$$t = C[t_1, \dots, t_n] \rightarrow_{\mathcal{B}_2}^* C[q_1, \dots, q_n] \rightarrow_{\Delta_\varepsilon(\mathcal{A}_2, \mathcal{B}_1)}^* C[r_1, \dots, r_n] \rightarrow_{\mathcal{B}_1}^* q$$

Here C is a multi-hole context with $n \geq 0$ holes. Using Lemma 9 we obtain ground terms u_1, \dots, u_n such that $u_i \rightarrow_{\mathcal{A}_2}^* q_i$ and $u \rightarrow_{\mathcal{B}_1}^* r_i$ for all $1 \leq i \leq n$. Define the term $u = C[u_1, \dots, u_n]$. We have $u \rightarrow_{\mathcal{B}_1}^* C[r_1, \dots, r_n] \rightarrow_{\mathcal{B}_1}^* q$ and thus $(s, u) \in L(\mathcal{P})$. Furthermore, $u \rightarrow_{\mathcal{A}_2}^* C[q_1, \dots, q_n]$ and thus also $(u, t) \in L(\mathcal{G})$. Hence $(s, t) \in L(\mathcal{P}) \circ L(\mathcal{G})$.

For the converse direction, let $(s, t) \in L(\mathcal{P})$ and $(t, u) \in L(\mathcal{G})$. So $s \rightarrow_{\mathcal{A}_1}^* p$ and $t \rightarrow_{\mathcal{B}_1}^* q$ with $(p, q) \in Q$. Moreover, there exists a multi-hole context C with $n \geq 0$ holes, terms $t_1, \dots, t_n, u_1, \dots, u_n$, and states r_1, \dots, r_n such that $t = C[t_1, \dots, t_n]$, $u = C[u_1, \dots, u_n]$, and $t_i \rightarrow_{\mathcal{A}_2}^* r_i$ and $u_i \rightarrow_{\mathcal{B}_2}^* r_i$ for all $1 \leq i \leq n$. The sequence $t \rightarrow_{\mathcal{B}_1}^* q$ can be written as $t = C[t_1, \dots, t_n] \rightarrow_{\mathcal{B}_1}^* C[q_1, \dots, q_n] \rightarrow_{\mathcal{B}_1}^* q$ for some states q_1, \dots, q_n . By Lemma 9, $r_i \rightarrow q_i$ is a transition rule in $\Delta_\varepsilon(\mathcal{A}_2, \mathcal{B}_1)$. Hence $u = C[u_1, \dots, u_n] \rightarrow_{\mathcal{B}_2}^* C[r_1, \dots, r_n] \rightarrow_{\Delta_\varepsilon(\mathcal{A}_2, \mathcal{B}_1)}^* C[q_1, \dots, q_n] \rightarrow_{\mathcal{B}_1}^* q$ and thus $(s, u) \in L(\mathcal{P}')$ as desired. □

Example 14 We consider the pair automaton $\mathcal{P}_{\mathcal{R}}$ and the GTT $\mathcal{G}_{\mathcal{R}}$ of Example 10. The construction in the above proof requires that $\mathcal{P}_{\mathcal{R}}$ and $\mathcal{G}_{\mathcal{R}}$ do not share states, so we rename the states of $\mathcal{G}_{\mathcal{R}}$ (by adding a prime). We obtain the pair automaton $\mathcal{P}' = ((1, 4), (2, 4), (3, 2)), \mathcal{A}', \mathcal{B}'$ with

$$\begin{array}{llll} \mathcal{A}': & a \rightarrow 0 & b \rightarrow 0 & f(0) \rightarrow 0 & g(0, 0) \rightarrow 0 \\ & a \rightarrow 1 & & f(1) \rightarrow 2 & g(1, 0) \rightarrow 3 \end{array}$$

$$\begin{array}{c}
 \frac{q \mathcal{A} \leftarrow p \quad p \rightsquigarrow r}{q \rightsquigarrow r} \text{ [a]} \qquad \frac{p \rightsquigarrow q \quad q \rightarrow_{\mathcal{B}} r}{p \rightsquigarrow r} \text{ [b]} \qquad \frac{p \rightsquigarrow q \quad q \rightsquigarrow r}{p \rightsquigarrow r} \text{ [t]} \\
 \frac{p \mathcal{A} \leftarrow f(p_1, \dots, p_n) \quad p_1 \rightsquigarrow q_1 \cdots p_n \rightsquigarrow q_n \quad f(q_1, \dots, q_n) \rightarrow_{\mathcal{B}} q}{p \rightsquigarrow q} \text{ [c]}
 \end{array}$$

Fig. 5 $\Delta_+(\mathcal{A}, \mathcal{B})$

$$\begin{array}{cccccc}
 \mathcal{B}': & a \rightarrow 1 & b \rightarrow 4 & f(1) \rightarrow 2 & 0' \rightarrow 1 & 1' \rightarrow 1 \\
 & a \rightarrow 5' & b \rightarrow 4' & f(5') \rightarrow 6' & 0' \rightarrow 4 & 0' \rightarrow 2 \\
 & 2' \rightarrow 2 & 4' \rightarrow 1 & 4' \rightarrow 2 & &
 \end{array}$$

We can also trim the resulting pair automata by trimming the underlying automata \mathcal{A}' and \mathcal{B}' . We declare a state q of \mathcal{A}' to be productive if $C[q] \rightarrow_{\mathcal{A}'}^* r$ for some context C and state $r \in \{p \mid (p, p') \in Q\}$. For the automaton \mathcal{B}' we use the second components $\{p' \mid (p, p') \in Q\}$. In our case \mathcal{A}' is already trim, but \mathcal{B}' simplifies to

$$a \rightarrow 1 \quad b \rightarrow 4 \quad f(1) \rightarrow 2 \quad b \rightarrow 4' \quad 4' \rightarrow 1 \quad 4' \rightarrow 2$$

We have $L(\mathcal{P}') = \{(f(a), b), (a, b)\} \cup \{g(a, t) \mid t \in \mathcal{T}(\mathcal{F})\} \times \{b, f(a), f(b)\}$, which indeed coincides with the relation $\rightarrow_{\varepsilon} \cdot \twoheadrightarrow$ induced by our leading TRS \mathcal{R} .

Theorem 7 ($A ::= A \widehat{\circ} A$) *Anchored GTT relations are effectively closed under modified composition.* ✔✔

Proof The construction $L(\mathcal{P}) \times L(\mathcal{G}) \mapsto L(\mathcal{P}')$ in the proof of Lemma 11 and its symmetric counterpart $L(\mathcal{G}) \times L(\mathcal{P}) \mapsto L(\mathcal{P}')$ in connection with Lemma 10 ensure that $\bowtie_1 \widehat{\circ} \bowtie_2$ is an anchored GTT relation. □

In Theorem 6 we have seen that anchored GTT relations are closed under transitive closure. GTT relations are also closed under transitive closure, which is the reason they were developed in the first place, but the construction is different from the one for anchored GTT relations and the correctness proof is considerably more involved. We present this construction as a modified transitive closure operation that preserves anchored GTT relations.

Definition 11 The modified transitive closure $\bowtie^{\hat{\top}}$ of a binary relation \bowtie on ground terms is defined as the relation

$$\bowtie^{\hat{\top}} = (\bowtie \overset{\geq}{\succ})^+ \circ \bowtie \circ (\bowtie \overset{\geq}{\succ})^+$$

We have $(\bowtie^{\hat{\top}}) \overset{\geq}{\succ} = (\bowtie \overset{\geq}{\succ})^+$. The proof that anchored GTT relations are effectively closed under $\hat{\top}$ employs the set $\Delta_+(\mathcal{A}, \mathcal{B})$ consisting of ε -transitions $p \rightsquigarrow q$ that are computed by the inference rules in Fig. 5.

Definition 12 Given a GTT $\mathcal{G} = (\mathcal{A}, \mathcal{B})$, we write A_+ for $\mathcal{A} \cup \Delta_+(\mathcal{B}, \mathcal{A})$ and B_+ for $\mathcal{B} \cup \Delta_+(\mathcal{A}, \mathcal{B})$. The GTT \mathcal{G}_+ is defined as $(\mathcal{A}_+, \mathcal{B}_+)$.

According to the following lemma, the multi-hole context closure of an anchored GTT relation is a GTT relation using the same GTT.

Lemma 12 *For every GTT \mathcal{G} , $L(\mathcal{G}) = L_a(\mathcal{G}) \overset{\geq}{\succ}$.* ✔✔

Proof Let $\mathcal{G} = (\mathcal{A}, \mathcal{B})$. If $(s, t) \in L(\mathcal{G})$ then there exist a context C with $n \geq 0$ holes, terms $s_1, \dots, s_n, t_1, \dots, t_n$, and states q_1, \dots, q_n with $s = C[s_1, \dots, s_n]$, $t = C[t_1, \dots, t_n]$, and $s_i \xrightarrow{*}_{\mathcal{A}} q_i \xleftarrow{*}_{\mathcal{B}} t_i$ for all $1 \leq i \leq n$. We have $(s_i, t_i) \in L_a(\mathcal{G})$ for all $1 \leq i \leq n$ by definition of anchored GTTs. Moreover, $C \in \mathcal{C}_{\geq} \cap \mathcal{C}_{\geq}^{\geq}$. Hence $(s, t) \in L_a(\mathcal{G})_{\geq}^{\geq}$. The converse is equally easy. \square

Lemma 13 *Let $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ be a GTT. If $(p, q) \in \Delta_+(\mathcal{A}, \mathcal{B})$ then $(s, t) \in L(\mathcal{G}^-)^+$ for some ground terms $s \in L(\mathcal{A}, p)$ and $t \in L(\mathcal{B}, q)$.* \checkmark

Proof We use induction on the relation \rightsquigarrow defined by the inference rules in Fig. 5. In the base case [c] is used with $p \xleftarrow{\mathcal{A}} a$ and $a \xrightarrow{\mathcal{B}} q$ for some constant a and hence we can take $s = t = a$. For the induction step we consider four cases, depending on which inference rule is used to derive $p \rightsquigarrow q$. First suppose [c] is used. So there exist transition rules $f(p_1, \dots, p_n) \rightarrow p$ in \mathcal{A} and $f(q_1, \dots, q_n) \rightarrow q$ in \mathcal{B} such that $p_i \rightsquigarrow q_i$ for $1 \leq i \leq n$. The induction hypothesis yields ground terms $s_1, \dots, s_n, t_1, \dots, t_n$ such that $(s_i, t_i) \in L(\mathcal{G}^-)^+$, $s_i \in L(\mathcal{A}, p_i)$, and $t_i \in L(\mathcal{B}, q_i)$ for $1 \leq i \leq n$. Let $s = f(s_1, \dots, s_n)$ and $t = f(t_1, \dots, t_n)$. We have $s \in L(\mathcal{A}, p)$ and $t \in L(\mathcal{B}, q)$. Moreover, $(s, t) \in L(\mathcal{G}^-)^+$ because the transitive closure of a parallel relation is parallel. Next suppose [a] is applied to derive $p \rightsquigarrow q$. So there exists a state p' such that $p \xleftarrow{\mathcal{A}} p' \rightsquigarrow q$. The induction hypothesis yields ground terms s and t such that $(s, t) \in L(\mathcal{G}^-)^+$, $s \in L(\mathcal{A}, p')$, and $t \in L(\mathcal{B}, q)$. Hence also $s \in L(\mathcal{A}, p)$. The reasoning for [b] is similar. The final case is the transitivity rule [t]. So $p \rightsquigarrow r$ and $r \rightsquigarrow q$ for some state r . The induction hypothesis yields terms s, t, u, v such that $(s, u), (v, t) \in L(\mathcal{G}^-)^+$, $s \in L(\mathcal{A}, p), u \in L(\mathcal{B}, r), v \in L(\mathcal{A}, r)$, and $t \in L(\mathcal{B}, q)$. From $u \in L(\mathcal{B}, r)$ and $v \in L(\mathcal{A}, r)$ we infer $(u, v) \in L(\mathcal{G}^-)$. Together with $(s, u), (v, t) \in L(\mathcal{G}^-)^+$, we obtain the desired $(s, t) \in L(\mathcal{G}^-)^+$. \square

Lemma 14 *Let $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ be a GTT. Let $\mathcal{G}_+ = (\mathcal{A}_+, \mathcal{B}_+)$. If $s \xrightarrow{*}_{\mathcal{A}_+} q$ then $t \xrightarrow{*}_{\mathcal{A}} q$ for some ground term t with $(s, t) \in L(\mathcal{G})^+$.* \checkmark

Proof We proceed by induction on the length of the reduction $s \xrightarrow{*}_{\mathcal{A}_+} p$. If the last step is an epsilon transition $q \rightarrow p$ then the induction hypothesis yields a ground term u with $(s, u) \in L(\mathcal{G})^+$ and $u \in L(\mathcal{A}, q)$. If $q \rightarrow p$ is a transition from \mathcal{A} then $u \in L(\mathcal{A}, p)$, and we conclude by letting $t = u$; otherwise, $q \rightarrow p$ must come from $\Delta_+(\mathcal{B}, \mathcal{A})$, and using Lemma 13 we obtain ground terms v and w with $v \in L(\mathcal{B}, q), w \in L(\mathcal{A}, p)$, and $(v, w) \in L(\mathcal{G})^+$. This implies $(u, v) \in L(\mathcal{G})$ and thus $(s, w) \in L(\mathcal{G})^+$ by transitivity. Letting $t = w$ gives the desired result. If the last step is not an ε -transition, then it must be a transition $f(p_1, \dots, p_n) \rightarrow p$ from \mathcal{A} , and we have $s = f(s_1, \dots, s_n)$ for suitable s_1, \dots, s_n . We apply the induction hypothesis to each argument position, resulting in t_1, \dots, t_n with $(s_i, t_i) \in L(\mathcal{G})^+$ and $t_i \in L(\mathcal{A}, p_i)$ for $1 \leq i \leq n$. Let $t = f(t_1, \dots, t_n)$. We have $t \in L(\mathcal{A}, p)$. Since $L(\mathcal{G})^+$ is transitive and closed under contexts, we obtain $(s, t) \in L(\mathcal{G})^*$. Since $L(\mathcal{G})$ is reflexive, we actually have $(s, t) \in L(\mathcal{G})^+$ as desired. \square

Lemma 15 *Let $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ be a GTT. If $\mathcal{G}_+ = (\mathcal{A}_+, \mathcal{B}_+)$ then $\Delta_\varepsilon(\mathcal{A}_+, \mathcal{B}_+) = \Delta_+(\mathcal{A}, \mathcal{B})$.* \checkmark

Proof We first show $\Delta_\varepsilon(\mathcal{A}_+, \mathcal{B}_+) \subseteq \Delta_+(\mathcal{A}, \mathcal{B})$ via induction on the relation \rightsquigarrow defined by the inference rules in Fig. 3. We proceed by case analysis, so assume $(p, q) \in \Delta_\varepsilon(\mathcal{A}_+, \mathcal{B}_+)$ is derived from a congruence step [c]. Hence we obtain $(p, q) \in \Delta_+(\mathcal{A}, \mathcal{B})$ by a congruence step [c] of Fig. 5, the fact that the constructions only add ε -transitions, and the induction hypothesis. Next assume that we derived $(q, r) \in \Delta_\varepsilon(\mathcal{A}_+, \mathcal{B}_+)$ by an ε -step [a]. So $p \xrightarrow{\mathcal{A}_+} q$

and $p \rightsquigarrow r$. We have $\mathcal{A}_+ = \mathcal{A} \cup \Delta_+(\mathcal{B}, \mathcal{A})$. The result trivially follows for $p \rightarrow_{\mathcal{A}} q$. So let $(p, q) \in \Delta_+(\mathcal{B}, \mathcal{A})$. Hence $(q, p) \in \Delta_+(\mathcal{A}, \mathcal{B})$. The induction hypothesis yields $(p, r) \in \Delta_+(\mathcal{A}, \mathcal{B})$ and therefore $(q, r) \in \Delta_+(\mathcal{A}, \mathcal{B})$ using the transitivity rule [t]. The ε -step [b] case is obtained in the same way.

For the reverse inclusion we use induction on the relation \rightsquigarrow defined by the inference rules in Fig. 5 and argue in a similar fashion. Hence $\Delta_\varepsilon(\mathcal{A}_+, \mathcal{B}_+) = \Delta_+(\mathcal{A}, \mathcal{B})$ as desired. \square

Theorem 8 ($A ::= A^\dagger$) *Anchored GTT relations are effectively closed under modified transitive closure.* ✔✔

Proof Let $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ be a GTT. We show $L_a(\mathcal{G}_+) = L_a(\mathcal{G})^\dagger$. First let $(s, t) \in L_a(\mathcal{G}_+)$. So there exists a state q such that $s \rightarrow_{\mathcal{A}_+}^* q$ and $t \rightarrow_{\mathcal{B}_+}^* q$. Lemma 14 yields a ground term u such that $u \rightarrow_{\mathcal{A}}^* q$ and $(s, u) \in L(\mathcal{G})^+$. Applied to $\mathcal{G}^- = (\mathcal{B}, \mathcal{A})$, Lemma 14 yields a ground term v such that $v \rightarrow_{\mathcal{B}}^* q$ and $(t, v) \in L(\mathcal{G}^-)^+$. Hence $(u, v) \in L_a(\mathcal{G})$ and $(v, t) \in L(\mathcal{G})^+$. Consequently, $(s, t) \in L(\mathcal{G})^+ \circ L_a(\mathcal{G}) \circ L(\mathcal{G})^+$ and, using Lemma 12, $L(\mathcal{G})^+ \circ L_a(\mathcal{G}) \circ L(\mathcal{G})^+ = L_a(\mathcal{G})^\dagger$.

For the other direction we apply the modified composition operation $\widehat{\circ}$ of Definition 10 with $\triangleright_1 = \triangleright_2 = L_a(\mathcal{G}_+)$ and obtain

$$L_a(\mathcal{G}_+) \circ L(\mathcal{G}_+) \cup L(\mathcal{G}_+) \circ L_a(\mathcal{G}_+) \subseteq L_a(\mathcal{G}_+) \widehat{\circ} L_a(\mathcal{G}_+) = L_a(\mathcal{G}_+)$$

with the help of Lemma 15. Note that we do not get equality, as one direction in the proof of Lemma 11 requires disjoint state sets. Since $L_a(\mathcal{G}) \subseteq L_a(\mathcal{G}_+)$ we also have

$$L_a(\mathcal{G}) \circ L(\mathcal{G}_+) \cup L(\mathcal{G}_+) \circ L_a(\mathcal{G}) \subseteq L_a(\mathcal{G}_+)$$

At this point we can use the following well-known result in Kleene algebra

$$A \subseteq X \wedge B \circ X \subseteq X \wedge X \circ C \subseteq X \implies B^* \circ A \circ C^* \subseteq X$$

with $A = L_a(\mathcal{G})$, $B = C = L(\mathcal{G})$, and $X = L_a(\mathcal{G}_+)$. Since $L(\mathcal{G})^* = L(\mathcal{G})^+$, we are done. \square

Example 15 For the GTT $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ of Example 11 we have $\Delta_+(\mathcal{A}, \mathcal{B}) = \Delta_\varepsilon(\mathcal{A}, \mathcal{B})$. Hence $\mathcal{G}_+ = (\mathcal{A}_+, \mathcal{B}_+)$ adds the pairs of $\Delta_+(\mathcal{B}, \mathcal{A}) = \{(5, 0), (5, 1), (4, 0), (6, 0), (6, 2), (5, 4), (6, 4)\}$ as ε -transitions to \mathcal{A} and those of $\Delta_+(\mathcal{A}, \mathcal{B}) = \Delta_+(\mathcal{B}, \mathcal{A})^-$ to \mathcal{B} . We have $(g(a, b), f(b)) \in L_a(\mathcal{G}_+)$ as $g(a, b) \rightarrow_{\mathcal{A}_+}^* 6$ and $f(b) \rightarrow_{\mathcal{B}_+} f(4) \rightarrow_{\mathcal{B}_+} f(5) \rightarrow_{\mathcal{B}_+} 6$. The term pair $(f(a), f(b))$ does not belong to $L_a(\mathcal{G}_+)$.

The penultimate operation on anchored GTT relations that we consider is complement. This requires the determinization of pair automata.

Lemma 16 *For every pair automaton $\mathcal{P} = (Q, \mathcal{A}, \mathcal{B})$ there exist deterministic tree automata \mathcal{A}' and \mathcal{B}' and a binary relation Q^d such that $L(\mathcal{P}) = L((Q^d, \mathcal{A}', \mathcal{B}'))$.* ✔

Proof We use the subset construction to determinize \mathcal{A} and \mathcal{B} into equivalent deterministic tree automata \mathcal{A}' and \mathcal{B}' . As the binary state relation we take $Q^d = \{(X, Y) \mid (p, q) \in Q \text{ for some } p \in X \subseteq Q_A \text{ and } q \in Y \subseteq Q_B\}$. We have $L(\mathcal{P}) = L((Q^d, \mathcal{A}', \mathcal{B}'))$ by the correctness of the subset construction. \square

Theorem 9 ($A ::= A^c$) *Anchored GTT relations are effectively closed under complement.* ✔

Proof Let \mathcal{G} be an anchored GTT. According to Lemma 16 we may assume that $L(\mathcal{G})$ is accepted by a deterministic pair automaton $\mathcal{P} = (Q, \mathcal{A}, \mathcal{B})$. Without loss of generality we may further assume that \mathcal{A} and \mathcal{B} are completely defined. It follows that $L(\mathcal{P})^c = (Q^c, \mathcal{A}, \mathcal{B})$ where $Q^c = (Q_A \times Q_B) \setminus Q$. \square

It is worth noting that GTT relations are *not* closed under complement [8, Exercise 3.4].

Example 16 For the pair automaton $\mathcal{P}_R = (Q, \mathcal{A}, \mathcal{B})$ of Example 10 we have $Q = \{(1, 4), (2, 4), (3, 2)\}$. Determinizing \mathcal{A} yields the tree automaton \mathcal{A}' with the following transition rules:

$$a \rightarrow A \quad b \rightarrow B \quad f(X) \rightarrow \begin{cases} C & \text{if } X = A \\ B & \text{otherwise} \end{cases} \quad g(X, Y) \rightarrow \begin{cases} D & \text{if } X = A \\ B & \text{otherwise} \end{cases}$$

for all $X, Y \in \{A, B, C, D\}$. Here $A = \{0, 1\}$, $B = \{0\}$, $C = \{0, 2\}$, and $D = \{0, 3\}$. Next we determinize \mathcal{B} to obtain the tree automaton \mathcal{B}' consisting of the following transition rules:

$$a \rightarrow E \quad b \rightarrow F \quad f(X) \rightarrow \begin{cases} G & \text{if } X = E \\ H & \text{otherwise} \end{cases} \quad g(X, Y) \rightarrow H$$

for all $X, Y \in \{E, F, G, H\}$. Here $E = \{1\}$, $F = \{4\}$, $G = \{2\}$, and $H = \emptyset$. The transition rules for g are added to make \mathcal{B}' completely defined. Now the complement $L(\mathcal{G})^c$ of $L(\mathcal{G})$ is accepted by the pair automaton $(Q', \mathcal{A}', \mathcal{B}')$ with

$$Q' = (\{A, B, C, D\} \times \{E, F, G, H\}) \setminus \{(A, F), (C, F), (D, G)\}$$

The final closure property of anchored GTT relations that we mention is intersection.

Lemma 17 ($A ::= A \cap A$) *Anchored GTT relations are effectively closed under intersection.*

Proof This follows from Theorem 9 and Lemma 10. □

The formalized proof uses a more efficient product construction, to avoid the subset construction of the complement.

5.3 Regular Relations

We continue with operations on regular relations. Again, most of the results and constructions are known. We provide detailed proofs that form the basis of the formalization. The following lemma takes care of transforming anchored GTT relations into binary regular (i.e., RR_2) relations.

Theorem 10 ($R ::= A$) *Every anchored GTT relation is an RR_2 relation.* ✓

Proof Let $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ be a GTT. We construct an RR_2 automaton that accepts $L_a(\mathcal{G})$. We use a product construction with states pq where p is a state of \mathcal{A} or \perp , and q is a state of \mathcal{B} or \perp ; the state $\perp\perp$ is not used. The transitions are

$$\begin{aligned} fg(p_1q_1, \dots, p_kq_k) &\rightarrow pq \\ f\perp(p_1\perp, \dots, p_n\perp) &\rightarrow p\perp \\ \perp g(\perp q_1, \dots, \perp q_m) &\rightarrow \perp q \end{aligned}$$

for all $f(p_1, \dots, p_n) \rightarrow p \in \mathcal{A}$ and $g(q_1, \dots, q_m) \rightarrow q \in \mathcal{B}$, where $k = \max(n, m)$ and $p_i = \perp$ if $n < i \leq k$ and $q_j = \perp$ if $m < j \leq k$, and

$$\begin{aligned} pq &\rightarrow p'q && \text{for all } p \rightarrow p' \in \mathcal{A} \text{ and } q \in Q_{\mathcal{B}} \cup \{\perp\} \\ pq &\rightarrow pq' && \text{for all } q \rightarrow q' \in \mathcal{B} \text{ and } p \in Q_{\mathcal{A}} \cup \{\perp\} \end{aligned}$$

These transitions accept $\langle s, t \rangle$ in state pq if and only if $s \in L(\mathcal{A}, p)$ and $t \in L(\mathcal{B}, q)$. As final states we pick pp with $p \in Q_{\mathcal{A}} \cap Q_{\mathcal{B}}$. A straightforward induction proof reveals that the resulting tree automaton accepts $L_a(\mathcal{G})$. \square

We illustrate the construction on our leading example.

Example 17 For the anchored GTT \mathcal{G} of Example 11 we obtain the RR_2 automaton $\mathcal{A} = (\mathcal{F}^{(2)}, Q, Q_f, \Delta)$ with $Q = (\{0, 1, 2, 3, 4, 6, \perp\} \times \{4, 5, 6, \perp\}) \setminus \{\perp\perp\}$, $Q_f = \{44, 66\}$, and Δ consisting of the following transition rules:

aa \rightarrow 05	ab \rightarrow 04	af(\perp 5) \rightarrow 06
aa \rightarrow 15	ab \rightarrow 14	af(\perp 5) \rightarrow 16
ba \rightarrow 05	bb \rightarrow 04	bf(\perp 5) \rightarrow 06
fa(0 \perp) \rightarrow 05	fb(0 \perp) \rightarrow 04	ff(05) \rightarrow 06
fa(1 \perp) \rightarrow 25	fb(1 \perp) \rightarrow 24	ff(15) \rightarrow 26
ga(0 \perp , 0 \perp) \rightarrow 05	gb(0 \perp , 0 \perp) \rightarrow 04	gf(05, 0 \perp) \rightarrow 06
ga(1 \perp , 0 \perp) \rightarrow 35	gb(1 \perp , 0 \perp) \rightarrow 34	gf(15, 0 \perp) \rightarrow 36
a \perp \rightarrow 0 \perp	b \perp \rightarrow 0 \perp	\perp a \rightarrow \perp 5
a \perp \rightarrow 1 \perp		\perp b \rightarrow \perp 4
f \perp (0 \perp) \rightarrow 0 \perp	f \perp (1 \perp) \rightarrow 2 \perp	\perp f(\perp 5) \rightarrow \perp 6
g \perp (0 \perp , 0 \perp) \rightarrow 0 \perp	g \perp (1 \perp , 0 \perp) \rightarrow 3 \perp	
14 \rightarrow 44	24 \rightarrow 44	34 \rightarrow 64
15 \rightarrow 45	25 \rightarrow 45	35 \rightarrow 65
16 \rightarrow 46	26 \rightarrow 46	36 \rightarrow 66
1 \perp \rightarrow 4 \perp	2 \perp \rightarrow 4 \perp	3 \perp \rightarrow 6 \perp

We have

$$\langle g(a, f(b)), f(a) \rangle = gf(aa, f\perp(b\perp)) \rightarrow_{\Delta}^* gf(15, f\perp(0\perp)) \rightarrow_{\Delta} gf(15, 0\perp) \rightarrow_{\Delta}^* 66$$

The various context closure operations are taken care of in the following general result.

Theorem 11 ($R ::= R_p^n$) If R is an RR_2 relation then R_p^n is an RR_2 relation, for all $n \in \{\geq, 1, >\}$ and $p \in \{\geq, \varepsilon, >\}$. ✔✔✔✔✔✔✔✔✔✔

Proof Let $\mathcal{A} = (\mathcal{F}^{(2)}, Q, Q_f, \Delta)$ be the RR_2 automaton that accepts R . We add two new states $*$ and \checkmark . In the former the encoding of the identity relation on ground terms will be accepted. The latter will serve as the unique final state (unless specified otherwise). This is achieved by extending Δ with the transitions $ff(*, \dots, *) \rightarrow *$ for every $f \in \mathcal{F}$ and $q \rightarrow \checkmark$ for every $q \in Q_f$. The resulting automaton $\mathcal{A}' = (\mathcal{F}^{(2)}, Q \cup \{*, \checkmark\}, \{\checkmark\}, \Delta')$ is equivalent to \mathcal{A} and the starting point for the various context closure operations.

- For $n = 1$ and $p = \geq$ we extend Δ with all rules of the form

$$ff(*, \dots, *, \checkmark, *, \dots, *) \rightarrow \checkmark$$

- For $p = >$ we need a new final state \checkmark' to ensure that the surrounding context is non-empty:

$$ff(*, \dots, *, \checkmark, *, \dots, *) \rightarrow \checkmark' \quad ff(*, \dots, *, \checkmark', *, \dots, *) \rightarrow \checkmark'$$

This is sufficient for $n = 1$. For $n = >$ we add the single ε -transition $\checkmark \rightarrow *$ and for $n = \geq$ we additionally add a new final state $*'$ together with transition rules ensuring that the accepted relation is reflexive:

$$ff(*', \dots, *') \rightarrow *'$$

- For $n = p = \geq$ we make $*$ the new (and only) final state and add the ε -transition $\checkmark \rightarrow *$.
- For $p = \varepsilon$ and $n \in \{1, >\}$ we have $R_p^n = R$ and thus we can just take the RR_2 automaton A . For $n = \geq$ we have $R_p^n = R^=$ and declare $*$ as an additional final state.
- In the remaining case we have $p = \geq$ and $n = >$. We extend Δ with all rules of the form

$$ff(*, \dots, *, \checkmark, *, \dots, *) \rightarrow \checkmark$$

and the single ε -transition $\checkmark \rightarrow *$.

The proof details can be found in the formalization. □

Example 18 The following transition rules are added to the RR_2 automaton of Example 17 to model the relation $L_\alpha(\mathcal{G})_{\geq}^= = \dot{\mapsto}_{>\varepsilon}$:

$aa \rightarrow *$	$44 \rightarrow \checkmark$	$ff(\checkmark) \rightarrow \checkmark'$	$ff(\checkmark') \rightarrow \checkmark'$
$bb \rightarrow *$	$66 \rightarrow \checkmark$	$gg(\checkmark, *) \rightarrow \checkmark'$	$gg(\checkmark', *) \rightarrow \checkmark'$
$ff(*) \rightarrow *$	$\checkmark \rightarrow *$	$gg(*, \checkmark) \rightarrow \checkmark'$	$gg(*, \checkmark') \rightarrow \checkmark'$
$gg(*, *) \rightarrow *$			

The encoding of the term pair $(g(f(a), f(a)), g(b, f(b)))$ is accepted: $gg(fb(a\perp), ff(ab)) \rightarrow^* gg(fb(1\perp), ff(14)) \rightarrow^* gg(24, ff(44)) \rightarrow^* gg(44, ff(\checkmark)) \rightarrow^* gg(\checkmark, \checkmark') \rightarrow gg(*, \checkmark') \rightarrow \checkmark'$.

We present one more operation that turns a regular set into an RR_2 relation. Here $=_T$ consists of all pairs (t, t) with $t \in T$.

Lemma 18 ($R ::= =_T$) *If $T \subseteq \mathcal{T}(\mathcal{F})$ is regular then $=_T$ is an RR_2 relation.* ✓

Proof Let $A = (\mathcal{F}, Q, Q_f, \Delta)$ be a tree automaton that accepts T . We turn A into the automaton $B = (\mathcal{F}^{(2)}, Q, Q_f, \Delta')$, where Δ' is obtained from Δ by modifying every transition rule $f(p_1, \dots, p_n) \rightarrow q$ of Δ into $ff(p_1, \dots, p_n) \rightarrow q$. The ε -transitions of Δ are kept. It is a trivial exercise to show that $L(B) = =_{L(A)} = =_T$. □

The following result is an immediate consequence of the corresponding closure properties on regular sets (Theorem 1).

Theorem 12 ($R ::= R \cup R \mid R \cap R$) *The class of n -ary regular relations is effectively closed under union and intersection for any $n \geq 0$.* ✓✓

The final closure operations on regular relations are required for the logical structure of formulas in the first-order theory of rewriting.

Theorem 13 ($R ::= R^c$) *The class of regular relations is effectively closed under complement.* ✓

Given a regular relation R , its complement is denoted by R^c . Note that $\langle R^c \rangle \neq \langle R \rangle^c$. The former is the topic of Theorem 13 and is used to model logical negation.

Proof Let $R \subseteq \mathcal{T}(\mathcal{F})^n$ be a regular relation. We have $\langle R^c \rangle = \langle R \rangle^c \setminus W^c$ where

$$W = \{t \in \mathcal{T}(\mathcal{F}^{(n)}) \mid t = \langle t_1, \dots, t_n \rangle \text{ for some } t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})\}$$

is the set of encodings of n -tuples of ground terms. It is not difficult to show that W is regular. The set $\langle R \rangle$ is regular by assumption. Hence the regularity of $\langle R^c \rangle$ is a consequence of Theorem 1. \square

Definition 13 Let R be an n -ary relation over $\mathcal{T}(\mathcal{F})$. If $1 \leq i \leq n + 1$ then the i -th cylindrification of R is the relation

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

Moreover, if σ is a permutation on $\{1, \dots, n\}$ then

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

Theorem 14 *The class of regular relations is effectively closed under cylindrification and permutation.* ✔✔

In [8, Proposition 3.2.12] the closure under cylindrification is obtained via an inverse homomorphic image, resulting in a shorter proof. The proof of the latter operates on completely defined deterministic tree automata. The (formalized) proof below operates on arbitrary tree automata.

Proof Let $\mathcal{A} = (\mathcal{F}^{(n)}, Q, Q_f, \Delta)$ be a tree automaton that accepts $\langle R \rangle$. We construct tree automata that accept $\langle C_i(R) \rangle$ and $\langle \sigma(R) \rangle$. We first consider permutation. Let σ be a permutation on $\{1, \dots, n\}$ and define $\mathcal{A}_\sigma = (\mathcal{F}^{(n)}, Q, Q_f, \Delta_\sigma)$ where Δ_σ is obtained from Δ by replacing every transition rule of the form $f_1 \cdots f_n(p_1, \dots, p_m) \rightarrow q$ with $f_{\sigma(1)} \cdots f_{\sigma(n)}(p_1, \dots, p_m) \rightarrow q$. Epsilon transitions in Δ are not affected. To conclude $L(\mathcal{A}_\sigma) = \langle \sigma(R) \rangle$, we first define the effect of σ on terms in $\mathcal{T}(\mathcal{F}^{(n)})$:

$$\sigma(t) = f_{\sigma(1)} \cdots f_{\sigma(n)}(\sigma(t_1), \dots, \sigma(t_m))$$

for $t = f_1 \cdots f_n(t_1, \dots, t_m)$. The following preliminary fact

$$\langle t_{\sigma(1)}, \dots, t_{\sigma(n)} \rangle = \sigma(\langle t_1, \dots, t_n \rangle) \tag{*_\sigma}$$

is proved as follows. We have

$$\text{Pos}(\langle t_{\sigma(1)}, \dots, t_{\sigma(n)} \rangle) = \text{Pos}(t_1) \cup \dots \cup \text{Pos}(t_n) = \text{Pos}(\langle t \rangle) = \text{Pos}(\sigma(\langle t \rangle))$$

and, for every position $p \in \text{Pos}(\langle t_{\sigma(1)}, \dots, t_{\sigma(n)} \rangle)$,

$$\langle t_{\sigma(1)}, \dots, t_{\sigma(n)} \rangle(p) = f_1 \cdots f_n = \sigma(\langle t \rangle)(p)$$

where $f_i = t_{\sigma(i)}(p)$ if $p \in \text{Pos}(t_{\sigma(i)})$ and $f_i = \perp$ otherwise. We now prove

$$\langle t_1, \dots, t_n \rangle \xrightarrow{*_{\mathcal{A}}} q \iff \langle t_{\sigma(1)}, \dots, t_{\sigma(n)} \rangle \xrightarrow{*_{\sigma(\mathcal{A})}} q \tag{(6)}$$

for all terms $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F} \cup \{\perp\})$ and states $q \in Q$. Suppose

$$\langle t_1, \dots, t_n \rangle = f_1 \cdots f_n(u_1, \dots, u_m) \xrightarrow{*_{\mathcal{A}}} q$$

So there exists a transition rule $f_1 \cdots f_n(q_1, \dots, q_m) \rightarrow p \in \Delta$ with $p \xrightarrow{*_{\mathcal{A}}} q$ and $u_i \xrightarrow{*_{\mathcal{A}}} q_i$ for all $1 \leq i \leq m$. We have $f_{\sigma(1)} \cdots f_{\sigma(n)}(q_1, \dots, q_m) \rightarrow p \in \Delta_\sigma$ and $p \xrightarrow{*_{\sigma(\mathcal{A})}} q$. Using $(*_\sigma)$ the induction hypothesis yields $\sigma(u_i) \xrightarrow{*_{\sigma(\mathcal{A})}} q_i$ for $1 \leq i \leq m$ and thus

$$\langle t_{\sigma(1)}, \dots, t_{\sigma(n)} \rangle = f_{\sigma(1)} \cdots f_{\sigma(n)}(\sigma(u_1), \dots, \sigma(u_m)) \xrightarrow{*_{\sigma(\mathcal{A})}} q$$

The converse is proved in a similar fashion. By specializing (6) to terms $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})$ and states $q \in \mathcal{Q}_f$ we obtain $L(\sigma(\mathcal{A})) = \{\sigma(\langle t_1, \dots, t_n \rangle) \mid \langle t_1, \dots, t_n \rangle \in L(\mathcal{A})\} = L(\langle \sigma(R) \rangle)$.

Next we consider cylindricfication. Let $i \in \{1, \dots, n + 1\}$. We define the tree automaton $\mathcal{A}_{C_i} = (\mathcal{F}^{(n+1)}, (\mathcal{Q} \cup \{\perp\}) \times \{\top, \perp\}, \mathcal{Q}_f \times \{\top\}, \Delta_{C_i})$ where \perp is a fresh state and Δ_{C_i} is obtained from Δ by replacing every transition rule of the form

$$f_1 \cdots f_{i-1} f_i \cdots f_n(p_1, \dots, p_m) \rightarrow q$$

with the transitions

$$\begin{aligned} f_1 \cdots f_{i-1} g f_i \cdots f_n(p_1 q_1, \dots, p_m q_m, \dots, p_k q_k) &\rightarrow q \top \\ f_1 \cdots f_{i-1} \perp f_i \cdots f_n(p_1 \perp, \dots, p_m \perp) &\rightarrow q \perp \end{aligned}$$

for all l -ary $g \in \mathcal{F}$. Here $k = \max(m, l)$ is the arity of $f_1 \cdots f_{i-1} g f_i \cdots f_n$. Moreover, $p_j = \perp$ for all $m < j \leq k$, and

$$q_j = \begin{cases} \top & \text{if } j \leq l \\ \perp & \text{if } j > l \end{cases}$$

for all $1 \leq j \leq k$. Additionally, Δ_{C_i} contains the transition rule

$$\perp \cdots \perp g \perp \cdots \perp (\perp \top, \dots, \perp \top) \rightarrow \perp \top$$

for every $g \in \mathcal{F}$. Here g is the i -th element in $\perp \cdots \perp g \perp \cdots \perp$. Finally, for every ε -transition $p \rightarrow q$ in Δ we add $p \top \rightarrow q \top$ and $p \perp \rightarrow q \perp$ to Δ_{C_i} . The purpose of the second component \perp/\top in states of \mathcal{A}_{C_i} is to mark whether states are reached by terms where (\top) the i -th position in the encoded tuple is a term in $\mathcal{T}(\mathcal{F})$, or (\perp) it is \perp . In order to show $L(\mathcal{A}_{C_i}) = \langle C_i(R) \rangle$, we simplify the notation by considering $i = 1$, which entails no loss of generality as regular relations are closed under permutation. Again, first we define the effect of C_1 on terms in $\mathcal{T}(\mathcal{F}^{(1)}) \times \mathcal{T}(\mathcal{F}^{(n)})$:

$$C_1(s, t) = f f_1 \cdots f_n(C_1(s_1, u_1), \dots, C_1(s_k, u_k))$$

for $s = f(s_1, \dots, s_l)$ and $t = f_1 \cdots f_n(u_1, \dots, u_m)$. Here $k = \max(l, m)$ is the arity of $f f_1 \cdots f_n$, $s_j = \perp$ for $l < j$ and $u_j = \perp^n$ for $m < j$. By induction on $s \in \mathcal{T}(\mathcal{F}^{(1)})$ and $t \in \mathcal{T}(\mathcal{F}^{(n)})$ we show the preliminary statements

$$\begin{aligned} \mathcal{Pos}(C_1(\perp, t)) = \mathcal{Pos}(t) \quad \text{and} \quad C_1(\perp, t)(p) = \perp t(p) \quad \text{for all } p \in \mathcal{Pos}(t) \quad (7) \\ \mathcal{Pos}(C_1(s, \perp^n)) = \mathcal{Pos}(s) \quad \text{and} \quad C_1(s, \perp^n)(p) = s(p) \perp^n \quad \text{for all } p \in \mathcal{Pos}(s) \quad (8) \end{aligned}$$

Let $t = f_1 \cdots f_n(u_1, \dots, u_m)$. We have $C_1(\perp, t) = \perp f_1 \cdots f_n(C_1(\perp, u_1), \dots, C_1(\perp, u_k))$ and obtain $\mathcal{Pos}(C_1(\perp, u_i)) = \mathcal{Pos}(u_i)$ and $C_1(\perp, u_i)(q) = \perp u_i(q)$ for all i $p \in \mathcal{Pos}(t)$ from the induction hypothesis. Note that i $p \in \mathcal{Pos}(t)$ if and only if $p \in \mathcal{Pos}(u_i)$. For $p = \varepsilon$ we have $C_1(\perp, t)(p) = \perp f_1 \cdots f_n = \perp t(p)$. This establishes (7). The proof of (8) is similar and omitted. These statements are used to prove $\mathcal{Pos}(C_1(s, t)) = \mathcal{Pos}(s) \cup \mathcal{Pos}(t)$ and $C_1(s, t)(p) = s(p)t(p)$ for all $p \in \mathcal{Pos}(s) \cup \mathcal{Pos}(t)$, by induction on $|s| + |t|$. Let $s = f(s_1, \dots, s_l)$ and $t = f_1 \cdots f_n(u_1, \dots, u_m)$. Let $k = \max(l, m)$ be the arity of $f f_1 \cdots f_n$. We have $\mathcal{Pos}(C_1(s_i, u_i)) = \mathcal{Pos}(s_i) \cup \mathcal{Pos}(u_i)$ and $C_1(s_i, u_i)(p) = s_i(p)u_i(p)$ for all $p \in \mathcal{Pos}(s_i) \cup \mathcal{Pos}(u_i)$ for all $1 \leq i \leq k$. For $i \leq \min(l, m)$ this follows from the induction hypothesis and for $i > \min(l, m)$ this follows from (7) or (8). Moreover,

$C_1(s, t)(\varepsilon) = ff_1 \cdots f_n = s(\varepsilon)t(\varepsilon)$ so the second statement also holds for $p = \varepsilon$. From these statements we immediately obtain

$$C_1(s, t) = \langle s, t_1, \dots, t_n \rangle \tag{*C}$$

for all terms $s \in \mathcal{T}(\mathcal{F}^{(1)})$ and $t = \langle t_1, \dots, t_n \rangle \in \mathcal{T}(\mathcal{F}^{(n)})$. The following two properties are easily proved by induction:

$$C_1(s, \perp^n) \rightarrow_{C_1(\mathcal{A})}^* \perp \top \tag{9}$$

for all terms $s \in \mathcal{T}(\mathcal{F})$ and

$$t \rightarrow_{\mathcal{A}}^* q \iff C_1(\perp, t) \rightarrow_{C_1(\mathcal{A})}^* q \perp \tag{10}$$

for all terms $t \in \mathcal{T}(\mathcal{F}^{(n)})$. For the first one we use induction on $s = f(s_1, \dots, s_l)$. We have $C_1(s, \perp^n) = f \perp^n (C_1(s_1, \perp^n), \dots, C_1(s_l, \perp^n))$ and obtain $C_1(s_i, \perp^n) \rightarrow_{C_1(\mathcal{A})}^* \perp \top$ for $1 \leq i \leq n$ from the induction hypothesis. By construction $f \perp^n (\perp \top, \dots, \perp \top) \rightarrow \perp \top \in \Delta_{C_1}$. Hence $C_1(s, \perp^n) \rightarrow_{C_1(\mathcal{A})}^* \perp \top$. The second property is proved by induction on $t = f_1 \cdots f_n(u_1, \dots, u_m)$. We have $C_1(\perp, t) = \perp f_1 \cdots f_n(C_1(\perp, u_1), \dots, C_1(\perp, u_m))$. First assume $t \rightarrow_{\mathcal{A}}^* q$. So there exists a transition rule $f_1 \cdots f_n(q_1, \dots, q_m) \rightarrow p \in \Delta$ with $p \rightarrow_{\mathcal{A}}^* q$ and $u_i \rightarrow_{\mathcal{A}}^* q_i$ for all $1 \leq i \leq m$. The induction hypothesis yields $C_1(\perp, u_i) \rightarrow_{C_1(\mathcal{A})}^* q_i \perp$ for $1 \leq i \leq m$. By construction $\perp f_1 \cdots f_n(q_1 \perp, \dots, q_m \perp) \rightarrow p \perp \in \Delta_{C_1}$ and $p \perp \rightarrow_{C_1(\mathcal{A})}^* q \perp$. Combining all this yields $C_1(\perp, t) \rightarrow_{C_1(\mathcal{A})}^* q \perp$. For the converse, assume $C_1(\perp, t) \rightarrow_{C_1(\mathcal{A})}^* q \perp$. So there exists a rule $\perp f_1 \cdots f_n(q_1 \perp, \dots, q_m \perp) \rightarrow p \perp \in \Delta_{C_1}$ with $p \perp \rightarrow_{C_1(\mathcal{A})}^* q \perp$ and $C_1(\perp, u_i) \rightarrow_{C_1(\mathcal{A})}^* q_i \perp$ for all $1 \leq i \leq m$. The induction hypothesis yields $u_i \rightarrow_{\mathcal{A}}^* q_i$ for $1 \leq i \leq m$. Furthermore, the transition rule $\perp f_1 \cdots f_n(q_1 \perp, \dots, q_m \perp) \rightarrow p \perp$ originates from $f_1 \cdots f_n(q_1, \dots, q_m) \rightarrow p \in \Delta$ and we obtain $p \perp \rightarrow_{C_1(\mathcal{A})}^* q \perp$ from $p \rightarrow_{\mathcal{A}}^* q$. Hence $t \rightarrow_{\mathcal{A}}^* q$ as desired. This completes the proofs of (9) and (10). Next we prove

$$t \rightarrow_{\mathcal{A}}^* q \iff C_1(s, t) \rightarrow_{C_1(\mathcal{A})}^* q \top \tag{11}$$

for all $s \in \mathcal{T}(\mathcal{F})$, $t \in \mathcal{T}(\mathcal{F}^{(n)})$ and $q \in Q$. For the only-if direction we use induction on $t = f_1 \cdots f_n(u_1, \dots, u_m)$. Let $s = f(s_1, \dots, s_l)$. From $t \rightarrow_{\mathcal{A}}^* q$ we obtain $f_1 \cdots f_n(p_1, \dots, p_m) \rightarrow p \in \Delta$ with $p \rightarrow_{\mathcal{A}}^* q$ and $u_i \rightarrow_{\mathcal{A}}^* p_i$ for all $1 \leq i \leq m$. We have

$$ff_1 \cdots f_n(p_1 q_1, \dots, p_m q_m, \dots, p_k q_k) \rightarrow p \top \in \Delta_{C_1}$$

by construction. Here $k = \max(l, m)$ is the arity of $ff_1 \cdots f_n$, $p_i = \perp$ for all $m < i \leq k$, $q_i = \top$ if $1 \leq i \leq l$ and $q_i = \perp$ if $l < i \leq k$. We have $p \top \rightarrow_{C_1(\mathcal{A})}^* q \top$ and $C_1(s, t) = ff_1 \cdots f_n(C_1(s_1, u_1), \dots, C_1(s_k, u_k))$ with $s_i = \perp$ for $l < i \leq k$ and $u_i = \perp^n$ for $m < i \leq k$. The induction hypothesis yields $C_1(s_i, u_i) \rightarrow_{C_1(\mathcal{A})}^* p_i \top$ for all $1 \leq i \leq \min(l, m)$. Note that $\top = q_i$. For $\min(l, m) < i \leq k$ we distinguish two cases.

- If $\min(l, m) = m$ then $m < i$ and thus $u_i = \perp^n$. We obtain $C_1(s_i, u_i) \rightarrow_{C_1(\mathcal{A})}^* \perp \top$ from (9). Note that $p_i = \perp$ and $q_i = \top$.
- If $\min(l, m) = l$ then $l < i$ and thus $s_i = \perp$. We obtain $C_1(s_i, u_i) \rightarrow_{C_1(\mathcal{A})}^* p_i \perp$ from (10). Note that $q_i = \perp$.

So in all cases we have $C_1(s_i, u_i) \rightarrow_{C_1(\mathcal{A})}^* p_i q_i$. Hence

$$C_1(s, t) \rightarrow_{C_1(\mathcal{A})}^* ff_1 \cdots f_n(p_1 q_1, \dots, p_m q_m, \dots, p_k q_k) \rightarrow_{C_1(\mathcal{A})} p \top \rightarrow_{C_1(\mathcal{A})}^* q \top$$

as desired. The if-direction of (11) is proved in a similar fashion. From

$$C_1(s, t) = ff_1 \cdots f_n(C_1(s_1, u_1), \dots, C_1(s_k, u_k)) \rightarrow_{C_1(\mathcal{A})}^* q\top$$

we obtain a rule $ff_1 \cdots f_n(p_1q_1, \dots, p_mq_m, \dots, p_kq_k) \rightarrow p\top \in \Delta_{C_1}$ with $p\top \rightarrow_{C_1(\mathcal{A})}^* q\top$ and $C_1(s_i, u_i) \rightarrow_{C_1(\mathcal{A})}^* p_iq_i$ for $1 \leq i \leq k$. We have $f_1 \cdots f_n(p_1, \dots, p_m) \rightarrow p \in \Delta$ and $p \rightarrow_{\mathcal{A}}^* q$ due to the construction of Δ_{C_1} . The induction hypothesis yields $u_i \rightarrow_{\mathcal{A}}^* p_i$ for $1 \leq i \leq m$ and thus $t = f_1 \cdots f_n(u_1, \dots, u_m) \rightarrow_{\mathcal{A}}^* q$. Specializing (11) to terms $t = \langle t_1, \dots, t_n \rangle$ with $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})$ and $q \in Q_f$ yields $L(C_1(\mathcal{A})) = \{ \langle s, t_1, \dots, t_n \rangle \mid \langle t_1, \dots, t_n \rangle \in L(\mathcal{A}) \text{ and } s \in \mathcal{T}(\mathcal{F}) \} = \langle C_1(R) \rangle$. \square

Note that for every RR_2 relation R , its inverse R^- is the same as $\sigma(R)$ for the permutation $\sigma = (12)$.

Corollary 2 ($R ::= R^-$) *The class of binary regular relations is effectively closed under inverse.* \square

Example 19 Consider the RR_2 automaton $\mathcal{A} = (\mathcal{F}^{(2)}, Q, Q_f, \Delta)$ of Example 17. We compute $C_2(\{(s, t, u) \mid s \rightarrow_\varepsilon u \text{ and } t \in \mathcal{T}(\mathcal{F})\})$. To this end, we transform \mathcal{A} by the construction in the above proof. This results in an automaton $\mathcal{B} = (\mathcal{F}^{(3)}, Q', Q'_f, \Delta')$ with $Q' = (Q \cup \{\perp\}) \times \{\top, \perp\}$, $Q'_f = \{44\top, 66\top\}$, and Δ' consisting of 183 transitions. Every non- ε -transition in Δ gives rise to five transitions in Δ' . For instance, the transitions

$$\begin{array}{lll} \text{aaa} \rightarrow 05\top & \text{afa}(\perp\top) \rightarrow 05\top & \text{aga}(\perp\top, \perp\top) \rightarrow 05\top \\ \text{aba} \rightarrow 05\top & \text{a}\perp\text{a} \rightarrow 05\perp & \end{array}$$

originate from $\text{aa} \rightarrow 05$ and the transitions

$$\begin{array}{lll} \perp\text{af}(\perp 5\top) \rightarrow \perp 6\top & \perp\text{ff}(\perp 5\top) \rightarrow \perp 6\top & \perp\text{gf}(\perp 5\top, \perp\top) \rightarrow \perp 6\top \\ \perp\text{bf}(\perp 5\top) \rightarrow \perp 6\top & \perp\perp\text{f}(\perp 5\perp) \rightarrow \perp 6\perp & \end{array}$$

originate from $\perp\text{f}(\perp 5) \rightarrow \perp 6$. Moreover, every ε -transition in Δ is duplicated in Δ' . For instance, $25 \rightarrow 45$ gives rise to $25\top \rightarrow 45\top$ and $25\perp \rightarrow 45\perp$. Finally, Δ' contains the transitions

$$\perp\text{a}\perp \rightarrow \perp\top \quad \perp\text{b}\perp \rightarrow \perp\top \quad \perp\text{f}\perp(\perp\top) \rightarrow \perp\top \quad \perp\text{g}\perp(\perp\top, \perp\top) \rightarrow \perp\top$$

So in total there are $31 \times 5 + 12 \times 2 + 4 = 183$ transitions in Δ' .

In Theorem 14 and its proof we have finally introduced all concepts needed to complete the proof that RR_n relations are closed under projection (Theorem 2). It remains to be shown that $L(\mathcal{A}_{\Pi_i}) = \langle \Pi_i(R) \rangle$.

Proof of Theorem 2 (cont'd) To simplify the notation, we consider Π_1 (which entails no loss of generality as regular relations are closed under permutation). Again, first we define the effect of Π_1 on terms in $\mathcal{T}(\mathcal{F}^{(n)})$:

$$\Pi_1(t) = f_2 \cdots f_n(\Pi_1(u_1), \dots, \Pi_1(u_k))$$

for $t = f_1 \cdots f_n(u_1, \dots, u_m)$. Here $k \leq m$ is the arity of $f_2 \cdots f_n$. We show

$$\Pi_1(C_1(s, t)) = t \tag{12}$$

for all terms $s \in \mathcal{T}(\mathcal{F}^{(1)})$ and $t \in \mathcal{T}(\mathcal{F}^{(n)})$ by induction on $|s| + |t|$. So let $s = f(s_1, \dots, s_l)$ and $t = f_1 \cdots f_n(u_1, \dots, u_m)$. We have

$$\begin{aligned} \Pi_1(C_1(s, t)) &= \Pi_1(ff_1 \cdots f_n(C_1(s_1, u_1), \dots, C_1(s_k, u_k))) \\ &= f_1 \cdots f_n(\Pi_1(C_1(s_1, u_1)), \dots, \Pi_1(C_1(s_m, u_m))) \\ &= f_1 \cdots f_n(u_1, \dots, u_m) = t \end{aligned}$$

Here $k = \max(l, m)$ is the arity of $ff_1 \cdots f_n$, $s_j = \perp$ for $l < j$, $u_j = \perp^n$ for $m < j$, and the induction hypothesis is applied to $\Pi_1(C_1(s_i, u_i))$ for $1 \leq i \leq m$. Now we can easily show

$$\Pi_1(\langle t_1, \dots, t_n \rangle) = \langle t_2, \dots, t_n \rangle \tag{*_{\Pi}}$$

for all terms $\langle t_1, \dots, t_n \rangle \in \mathcal{T}(\mathcal{F}^{(n)})$. From $(*_C)$ in the proof of Theorem 14 we obtain

$$\langle t_1, t_2, \dots, t_n \rangle = C_1(t_1, \langle t_2, \dots, t_n \rangle)$$

and thus $\Pi_1(\langle t_1, \dots, t_n \rangle) = \Pi_1(C_1(t_1, \langle t_2, \dots, t_n \rangle)) = \langle t_2, \dots, t_n \rangle$ using (12). We now prove the following two statements:

$$t \xrightarrow{*_{\mathcal{A}}} q \implies \Pi_1(t) \xrightarrow{*_{\Pi_1(\mathcal{A})}} q \tag{13}$$

for all terms $t \in \mathcal{T}(\mathcal{F}^{(n)})$ and states $q \in Q$, and

$$u \xrightarrow{*_{\Pi_1(\mathcal{A})}} q \implies t \xrightarrow{*_{\mathcal{A}}} q \text{ for some term } t \in \mathcal{T}(\mathcal{F}^{(n)}) \text{ with } \Pi_1(t) = u \tag{14}$$

for all terms $u \in \mathcal{T}(\mathcal{F}^{(n)})$. We prove the first statement by induction on t . Suppose

$$t = f_1 \cdots f_n(u_1, \dots, u_m) \xrightarrow{*_{\mathcal{A}}} q$$

So there exist a transition rule $f_1 \cdots f_n(q_1, \dots, q_m) \rightarrow p \in \Delta$ with $p \xrightarrow{*_{\mathcal{A}}} q$ such that $u_i \xrightarrow{*_{\mathcal{A}}} q_i$ for all $1 \leq i \leq m$. To simplify the reasoning, we assume that the condition $f_2 \cdots f_n \neq \perp^{n-1}$ in the definition of Δ_{Π_1} is temporarily lifted. This entails that $f_2 \cdots f_n(q_1, \dots, q_k) \rightarrow p$ is a transition rule in Δ_{Π_1} . Here $k \leq m$ is the arity of $f_2 \cdots f_n$. We have $p \xrightarrow{*_{\Pi_1(\mathcal{A})}} q$. The induction hypothesis yields $\Pi_1(u_i) \xrightarrow{*_{\Pi_1(\mathcal{A})}} q_i$ for $1 \leq i \leq m$. Hence

$$\Pi_1(t) = f_2 \cdots f_n(\Pi_1(u_1), \dots, \Pi_1(u_k)) \xrightarrow{*_{\Pi_1(\mathcal{A})}} f_2 \cdots f_n(q_1, \dots, q_k) \xrightarrow{*_{\Pi_1(\mathcal{A})}} q$$

as desired. For the second statement, suppose $u = f_2 \cdots f_n(u_1, \dots, u_k) \xrightarrow{*_{\Pi_1(\mathcal{A})}} q$ and so there exists a transition rule $f_2 \cdots f_n(q_1, \dots, q_k) \rightarrow p \in \Delta_{\Pi_1}$ with $p \xrightarrow{*_{\Pi_1(\mathcal{A})}} q$ and $u_i \xrightarrow{*_{\Pi_1(\mathcal{A})}} q_i$ for all $1 \leq i \leq k$. By construction of $\Pi_1(\mathcal{A})$, there exist a function symbol $f_1 \in \mathcal{F} \cup \{\perp\}$ and states q_{k+1}, \dots, q_m such that $f_1 f_2 \cdots f_n(q_1, \dots, q_m) \rightarrow p \in \Delta$. Here $m \geq k$ is the arity of $f_1 \cdots f_n$. From the induction hypothesis we obtain terms $v_1, \dots, v_k \in \mathcal{T}(\mathcal{F}^{(n)})$ such that $v_i \xrightarrow{*_{\mathcal{A}}} q_i$ and $\Pi_1(v_i) = u_i$ for $1 \leq i \leq k$. Because all states of \mathcal{A} are reachable, there exist terms $v_{k+1}, \dots, v_m \in \mathcal{T}(\mathcal{F}^{(n)})$ such that $v_j \xrightarrow{*_{\mathcal{A}}} q_j$ for $k+1 \leq j \leq m$. Now let $t = f_1 \cdots f_n(v_1, \dots, v_m)$. We clearly have $t \xrightarrow{*_{\mathcal{A}}} f_1 \cdots f_n(q_1, \dots, q_m) \xrightarrow{*_{\mathcal{A}}} p$. Moreover, $\Pi_1(t) = f_2 \cdots f_n(\Pi_1(v_1), \dots, \Pi_1(v_k)) = f_2 \cdots f_n(u_1, \dots, u_k) = u$. This concludes the proof of the two statements. Specializing statement (13) to $t = \langle t_1, \dots, t_n \rangle$ where $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F})$ and states $q \in Q_f$ yields $\Pi_1(L(\mathcal{A})) \subseteq L(\Pi_1(\mathcal{A}))$. From statement (14) we conclude $L(\Pi_1(\mathcal{A})) \subseteq \Pi_1(L(\mathcal{A}))$ and hence

$$L(\Pi_1(\mathcal{A})) = \{\Pi_1(\langle t_1, \dots, t_n \rangle) \mid \langle t_1, \dots, t_n \rangle \in L(\mathcal{A})\} = \langle \Pi_1(R) \rangle$$

It remains to show that the automaton $\Pi_1(\mathcal{A})$ does not use any rule $\perp^{n-1} \rightarrow p$ to accept terms when $n > 1$. Since $L(\Pi_1(\mathcal{A})) = \langle \Pi_1(R) \rangle$ and $\Pi_1(R) \subseteq \mathcal{T}(\mathcal{F})^{n-1}$, no term in $\langle \Pi_1(R) \rangle$ contains the function symbol \perp^{n-1} . □

5.4 Normal Form Predicate

At this point we have formalized proofs for the constructs in the grammar in Fig. 1, with the exception of the normal form predicate ($T ::= NF$). This predicate can be defined in the first-order theory of rewriting as

$$NF(t) \iff \neg \exists u (t \rightarrow u)$$

which gives rise to the following procedure:

1. Using Theorems 4, 10 and 11 an RR_2 automaton is constructed that accepts the encoding of the rewrite relation \rightarrow .
2. Using Theorem 2 the RR_2 automaton of step 1 is projected into a tree automaton that accepts the set of reducible ground terms, corresponding to the subformula $\exists u (t \rightarrow u)$.
3. Complementation (Theorem 13) is applied to the automaton of step 2 to obtain a tree automaton that accepts the set of ground normal forms.

Since projection may transform a deterministic tree automaton into a non-deterministic one, this is inefficient. In this section we provide a direct construction of a tree automaton that accepts the set of ground normal forms of a left-linear TRS, which goes back to Comon [6], and present a formalized correctness proof. Throughout this section \mathcal{R} is assumed to be left-linear.

We start with defining some preliminary concepts.

Definition 14 Given a signature \mathcal{F} , we write \mathcal{F}_\perp for the extension of \mathcal{F} with a fresh constant symbol \perp . Given $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, t^\perp denotes the result of replacing all variables in t by \perp :

$$x^\perp = \perp \qquad f(t_1, \dots, t_n)^\perp = f(t_1^\perp, \dots, t_n^\perp)$$

We define the partial order \leq on $\mathcal{T}(\mathcal{F}_\perp)$ as the least congruence that satisfies $\perp \leq t$ for all terms $t \in \mathcal{T}(\mathcal{F}_\perp)$:

$$\frac{}{\perp \leq t} \qquad \frac{t_1 \leq u_1 \quad \dots \quad t_n \leq u_n}{f(t_1, \dots, t_n) \leq f(u_1, \dots, u_n)}$$

The partial map $\uparrow: \mathcal{T}(\mathcal{F}_\perp) \times \mathcal{T}(\mathcal{F}_\perp) \rightarrow \mathcal{T}(\mathcal{F}_\perp)$ is defined as follows:

$$\perp \uparrow t = t \uparrow \perp = t \qquad f(t_1, \dots, t_n) \uparrow f(u_1, \dots, u_n) = f(t_1 \uparrow u_1, \dots, t_n \uparrow u_n)$$

It is not difficult to show that $t \uparrow u$ is the least upper bound of comparable terms t and u .

Definition 15 Let \mathcal{R} be a TRS over a signature \mathcal{F} . We write T^\perp for the set $\{t^\perp \mid t \triangleleft \ell \text{ for some } \ell \rightarrow r \in \mathcal{R}\} \cup \{\perp\}$. The set T_\uparrow is obtained by closing T^\perp under \uparrow .

Example 20 Consider the TRS \mathcal{R} consisting of following rules:

$$h(f(g(a), x, y)) \rightarrow g(a) \qquad g(f(x, h(x), y)) \rightarrow x \qquad h(f(x, y, h(a))) \rightarrow h(x)$$

We start by collecting the subterms of the left-hand sides:

$$T^\perp = \{\perp, a, g(a), h(\perp), h(a), f(g(a), \perp, \perp), f(\perp, h(\perp), \perp), f(\perp, \perp, h(a))\}$$

Closing T^\perp under \uparrow adds the following terms:

$$\begin{aligned} f(g(a), \perp, \perp) \uparrow f(\perp, h(\perp), \perp) &= f(g(a), h(\perp), \perp) \\ f(\perp, \perp, h(a)) \uparrow f(\perp, h(\perp), \perp) &= f(\perp, h(\perp), h(a)) \\ f(g(a), h(\perp), \perp) \uparrow f(\perp, h(\perp), h(a)) &= f(g(a), h(\perp), h(a)) \end{aligned}$$

Lemma 19 *The set T_{\uparrow} is finite.* ✔

Proof If $t \uparrow u$ is defined then $\text{Pos}(t \uparrow u) = \text{Pos}(t) \cup \text{Pos}(u)$. It follows that the positions of terms in $T_{\uparrow} \setminus T^{\perp}$ are positions of terms in T^{\perp} . Since T^{\perp} is finite, there are only finitely many such positions. Hence the finiteness of T_{\uparrow} follows from the finiteness of \mathcal{F} . □

Although the above proof is simple enough, we formalized the proof below which is based on a concrete algorithm to compute T_{\uparrow} . Actually, the algorithm presented below is based on a general saturation procedure, which is of independent interest.

Definition 16 Let $f : U \times U \rightarrow U$ be a (possibly partial) function and let S be a finite subset of U . The *closure* $C_f(S)$ is the least extension of S with the property that $f(a, b) \in C_f(S)$ whenever $a, b \in C_f(S)$ and $f(a, b)$ is defined.

The following lemma provides a sufficient condition for closures to exist. The proof gives a concrete algorithm to compute the closure.

Lemma 20 *If f is a total, associative, commutative, and idempotent function then $C_f(S)$ exists and is finite.* ✔

Proof If $S = \emptyset$ then $C_f(S) = \emptyset$ and the claim trivially holds. Suppose $S \neq \emptyset$ and let a be an arbitrary element in S . We show

$$C_f(S) = C_f(S \setminus \{a\}) \cup \{a\} \cup \{f(a, c) \mid c \in C_f(S \setminus \{a\})\}$$

Since S is finite, this gives rise to the following iterative algorithm to compute $C_f(S)$:

```

I := ∅;
for all x ∈ S do
    I := I ∪ {x} ∪ {f(x, y) | y ∈ I}
return I
    
```

In each iteration only finitely many elements are added. Hence $C_f(S)$ is finite. It remains to show the above equation. The inclusion from left to right is immediate from the definition of $C_f(S)$. Let b be an arbitrary element of $C_f(S)$. If $b \in S$ then $b \in C_f(S \setminus \{a\}) \cup \{a\}$. If $b \notin S$ then $b = f(a_1, f(a_2, \dots f(a_{n-1}, a_n) \dots))$ for some sequence of elements $a_1, \dots, a_n \in S$. If a is an element of this sequence then, using the properties of f , we may assume a appears exactly once in the sequence. Hence $b = f(a, c)$ for some element $c \in C_f(S \setminus \{a\})$. If a is not an element of a_1, \dots, a_n then $b \in C_f(S \setminus \{a\})$. This completes the proof. □

Since our function \uparrow is partial, we need to lift it to a total function that preserves associativity and commutativity. In our abstract setting this entails finding a binary predicate P on U such that $f(a, b)$ is defined if $P(a, b)$ holds. In addition, the following properties need to be fulfilled:

- P is reflexive and symmetric,
- if $P(a, f(b, c))$ and $P(b, c)$ hold then $P(a, b)$ and $P(f(a, b), c)$ hold as well, for all $a, b, c \in U$.

For the details we refer to the formalization.

Definition 17 The tree automaton $\mathcal{A}_{\text{NF}(\mathcal{R})} = (\mathcal{F}, Q, Q_f, \Delta)$ is defined as follows: $Q = Q_f = T_{\uparrow}$ and Δ consists of all transition rules $f(p_1, \dots, p_n) \rightarrow q$ such that $f(p_1, \dots, p_n)$ is no redex of \mathcal{R} and q is the maximal element of Q satisfying $q \leq f(p_1, \dots, p_n)$.³

³ Since states are terms from $T_{\uparrow} \subseteq \mathcal{T}(\mathcal{F}_{\perp})$ here, Definition 14 applies.

Example 21 For the TRS \mathcal{R} of Example 20, the tree automaton $\mathcal{A}_{\text{NF}(\mathcal{R})}$ consists of the following transition rules:

$$\begin{aligned}
 a &\rightarrow 1 & g(p) &\rightarrow \begin{cases} 2 & \text{if } p = 1 \\ 0 & \text{if } p \notin \{1, 6, 9, 10\} \end{cases} & h(p) &\rightarrow \begin{cases} 4 & \text{if } p = 1 \\ 3 & \text{if } p \notin \{1, 8, 10\} \end{cases} \\
 f(p, q, r) &\rightarrow \begin{cases} 5 & \text{if } p = 2, q \notin \{3, 4\} \\ 6 & \text{if } p \neq 2, q \in \{3, 4\}, r \neq 4 \\ 7 & \text{if } q \notin \{3, 4\}, r = 4 \\ 8 & \text{if } p = 2, q \in \{3, 4\}, r \neq 4 \end{cases} \\
 f(p, q, r) &\rightarrow \begin{cases} 9 & \text{if } p \neq 2, q \in \{3, 4\}, r = 4 \\ 10 & \text{if } p = 2, q \in \{3, 4\}, r = 4 \\ 0 & \text{otherwise} \end{cases}
 \end{aligned}$$

Here we use the following abbreviations:

$$\begin{aligned}
 0 &= \perp & 3 &= h(\perp) & 6 &= f(\perp, h(\perp), \perp) & 8 &= f(g(a), h(\perp), \perp) \\
 1 &= a & 4 &= h(a) & 7 &= f(\perp, \perp, h(a)) & 9 &= f(\perp, h(\perp), h(a)) \\
 2 &= g(a) & 5 &= f(g(a), \perp, \perp) & & & 10 &= f(g(a), h(\perp), h(a))
 \end{aligned}$$

As can be seen from the above example, the tree automaton $\mathcal{A}_{\text{NF}(\mathcal{R})}$ is not completely defined. Unlike the construction in [6], we do not have an additional state that is reached by all reducible ground terms.

Before proving that $\mathcal{A}_{\text{NF}(\mathcal{R})}$ accepts the ground normal forms of \mathcal{R} , we first show that $\mathcal{A}_{\text{NF}(\mathcal{R})}$ is well-defined, which amounts to showing that for every $f(p_1, \dots, p_n)$ with $f \in \mathcal{F}$ and $p_1, \dots, p_n \in T_\uparrow$ the set of states q such that $q \leq f(p_1, \dots, p_n)$ has a maximum element with respect to the partial order \leq .

Lemma 21 For every term $t \in \mathcal{T}(\mathcal{F}_\perp)$ the set $\{s \in T_\uparrow \mid s \leq t\}$ has a unique maximal element. ✔✔

Proof Let $S = \{s \in T_\uparrow \mid s \leq t\}$. Because $\perp \leq t$ and $\perp \in T_\uparrow$, $S \neq \emptyset$. If $s_1, s_2 \in S$ then $s_1 \leq t$ and $s_2 \leq t$ and thus $s_1 \uparrow s_2$ is defined and satisfies $s_1 \uparrow s_2 \leq t$. Since T_\uparrow is closed under \uparrow , $s_1 \uparrow s_2 \in T_\uparrow$ and thus $s_1 \uparrow s_2 \in S$. Consequently, S has a unique maximal element. \square

The next lemma is a trivial consequence of the fact that $\mathcal{A}_{\text{NF}(\mathcal{R})}$ has no ε -transitions.

Lemma 22 The tree automaton $\mathcal{A}_{\text{NF}(\mathcal{R})}$ is deterministic. ✔

Lemma 23 If $t \in \mathcal{T}(\mathcal{F})$ with $t \rightarrow_\Delta^* q$ and $s^\perp \leq t^\perp$ for a proper subterm s of some left-hand side of \mathcal{R} then $s^\perp \leq q$. ✔

Proof We use induction on t . Let $t = f(t_1, \dots, t_n)$. We have $t \rightarrow_\Delta^* f(q_1, \dots, q_n) \rightarrow_\Delta q$. We proceed by case analysis on s . If s is a variable then $s^\perp = \perp$ and, as \perp is minimal in \leq , we obtain $s^\perp \leq q$. Otherwise we must have $\text{root}(s) = f$ from the assumption $s^\perp \leq t^\perp$. So we may write $s = f(s_1, \dots, s_n)$. The induction hypothesis yields $s_i^\perp \leq q_i$ for all $1 \leq i \leq n$. Hence $s^\perp = f(s_1^\perp, \dots, s_n^\perp) \leq f(q_1, \dots, q_n)$. Additionally we have $s^\perp \in Q$ by Definition 17 as s is a proper subterm of a left-hand side of \mathcal{R} . Since $f(q_1, \dots, q_n) \rightarrow q$ is a transition rule, we obtain $f(s_1, \dots, s_n)^\perp \leq q$ from the maximality of q . \square

Table 1 Summary of (formalized) closure properties

Operation	GTTs	Anchored GTTs	RR ₂	Operation	Regular relations
Union	×	✓	✓	Union	✓
Intersection	×	✓	✓	Intersection	✓
Complement	×	✓	✓	Complement	✓
Composition	✓	✓*	<u>✓</u>	Projection	✓
Inverse	✓	✓	✓	Cylindrification	✓
Transitive closure	✓	✓*	×	Permutation	✓
Context closure	×	×	✓		

Using the previous result we can prove that no redex of \mathcal{R} reaches a state in $\mathcal{A}_{NF(\mathcal{R})}$.

Lemma 24 *If $t \in \mathcal{T}(\mathcal{F})$ is a redex then $t \rightarrow_{\Delta}^* q$ for no state $q \in T_{\uparrow}$.* ✓

Proof We have $\ell^{\perp} \leq t$ for some left-hand side ℓ of \mathcal{R} . For a proof by contradiction, assume $t \rightarrow_{\Delta}^* q$. Write $t = f(t_1, \dots, t_n)$. We have $t \rightarrow_{\Delta}^* f(q_1, \dots, q_n) \rightarrow_{\Delta} q$ and obtain $\ell^{\perp} \leq f(q_1, \dots, q_n)$ by a case analysis on ℓ and Lemma 23. Therefore the transition rule $f(q_1, \dots, q_n) \rightarrow_{\Delta} q$ cannot exist by Definition 17. □

Lemma 25 *If $t \rightarrow_{\Delta}^* q$ and $t \in \mathcal{T}(\mathcal{F})$ then $q \leq t$.* ✓

Proof We use induction on t . Let $t = f(t_1, \dots, t_n)$. We have $t \rightarrow_{\Delta}^* f(q_1, \dots, q_n) \rightarrow_{\Delta}^* q$. The induction hypothesis yields $q_i \leq t_i$ for all $1 \leq i \leq n$ and thus also $f(q_1, \dots, q_n) \leq f(t_1, \dots, t_n)$. We have $q \leq f(q_1, \dots, q_n)$ by Definition 17 and thus $q \leq t$ by the transitivity of \leq . □

Lemma 26 *If $t \in NF(\mathcal{R})$ then $t \rightarrow_{\Delta}^* q$ for some state $q \in T_{\uparrow}$.* ✓

Proof We use induction on t . Let $t = f(t_1, \dots, t_n)$. Since $t_1, \dots, t_n \in NF(\mathcal{R})$ we obtain $f(t_1, \dots, t_n) \rightarrow_{\Delta}^* f(q_1, \dots, q_n)$ from the induction hypothesis. Suppose $f(q_1, \dots, q_n)$ is a redex, so $\ell^{\perp} \leq f(q_1, \dots, q_n)$ for some left-hand side ℓ of \mathcal{R} . From Lemma 25 we obtain $q_i \leq t_i$ for all $1 \leq i \leq n$ and thus $f(q_1, \dots, q_n) \leq f(t_1, \dots, t_n)$. Hence $\ell^{\perp} \leq f(t_1, \dots, t_n)$. This however contradicts the assumption that t is a normal form. (Here we need left-linearity of \mathcal{R} .) Therefore $f(q_1, \dots, q_n)$ is no redex and thus, using Lemma 21, there exists a transition $f(q_1, \dots, q_n) \rightarrow q$ in Δ and thus $t \rightarrow_{\Delta}^* q$. □

Theorem 15 ($T ::= NF$) *If \mathcal{R} is a left-linear TRS then $L(\mathcal{A}_{NF(\mathcal{R})}) = NF(\mathcal{R})$.*

Proof Let $t \in \mathcal{T}(\mathcal{F})$. If $t \in NF(\mathcal{R})$ then $t \rightarrow_{\Delta}^* q$ for some state $q \in T_{\uparrow}$ by Lemma 26. Since all states in T_{\uparrow} are final, $t \in L(\mathcal{A}_{NF(\mathcal{R})})$. Next assume $t \notin NF(\mathcal{R})$. Hence $t = C[s]$ for some redex s . According to Lemma 24 s does not reach a state in $\mathcal{A}_{NF(\mathcal{R})}$. Hence also t cannot reach a state and thus $t \notin L(\mathcal{A}_{NF(\mathcal{R})})$. □

5.5 Decision Procedure

In Table 1 we summarize the effective closure properties that were presented in detail in this section and formalized in Isabelle. The asterisks indicate that for anchored GTTs we have two closure properties each. The underlined result (the closure of RR₂ relations under

Table 2 Binary predicates as RR_2 relations

$\rightarrow = (\rightarrow_\varepsilon) \supseteq^1$	$\leftarrow = ((\rightarrow_\varepsilon) \supseteq^1)^-$
$\rightarrow_\varepsilon = (\rightarrow_\varepsilon) \supseteq^1_\varepsilon$	$\rightarrow^+ = ((\rightarrow_\varepsilon) \hat{\supseteq}) \supseteq$
$\rightarrow_{>\varepsilon} = (\rightarrow_\varepsilon) \supseteq^1_{>}$	$\rightarrow^*_\varepsilon = ((\rightarrow_\varepsilon) \hat{\supseteq}) \supseteq^*_\varepsilon$
$\nrightarrow = (\rightarrow_\varepsilon) \supseteq^1_{\neq}$	$\rightarrow^* = ((\rightarrow_\varepsilon) \hat{\supseteq}) \supseteq^*_\neq$
$\rightarrow^+_e = ((\rightarrow_\varepsilon)^+) \supseteq^1_e$	$\leftarrow^* = (((\rightarrow_\varepsilon)^- \cup \rightarrow_\varepsilon) \hat{\supseteq}) \supseteq^*_\neq$
$\leftrightarrow = ((\rightarrow_\varepsilon)^- \cup \rightarrow_\varepsilon) \supseteq^1_{\leftrightarrow}$	$\downarrow = ((\rightarrow_\varepsilon) \hat{\supseteq} \circ (\rightarrow_\varepsilon)^- \hat{\supseteq}) \supseteq^1_{\downarrow}$
$\rightarrow^! = ((\rightarrow_\varepsilon) \hat{\supseteq}) \supseteq^1_{\rightarrow^!} \cap (\mathcal{T}(\mathcal{F}) \times \text{NF})$	

composition) is not used in the decision procedure but does hold: If R_1 and R_2 are RR_2 relations then $R_1 \circ R_2 = \Pi_2(C_3(R_1) \cap C_1(R_2))$. Concerning the empty entry in the table, it can be shown that GTT relations are closed under the context operation $(\cdot)_p^n$ if and only if $n \in \{\geq, 1, >\}$ and $p \in \{\geq, \varepsilon\}$. The second and third columns in the left part of Table 1 correspond to the A and R parts of the grammar in Fig. 1.

The logical structure of formulas in the first-order theory of rewriting is taken care of by the closure operations on regular relations listed in the second half of Table 1.

In Table 2 we show how some of the common binary predicates in term rewriting are represented as RR_2 relations using the corresponding operations. These are added to the language \mathcal{L} of the first-order theory of rewriting without compromising the decidability result that is presented below.

Theorem 16 *The first-order theory of rewriting is decidable for finite linear variable-separated TRSs.*

Proof Let $\varphi(x_1, \dots, x_n)$ be a first-order formula over the language \mathcal{L} with free variables x_1, \dots, x_n . Let \mathcal{R} be a finite linear variable-separated TRS over a signature \mathcal{F} . We construct an RR_n automaton that accepts the encoding of the relation $\llbracket \varphi \rrbracket = \{(t_1, \dots, t_n) \mid \mathcal{R} \models \varphi(t_1, \dots, t_n)\}$. For closed formulas, checking $\mathcal{R} \models \varphi$ then boils down to checking non-emptiness of $\llbracket \varphi \rrbracket$, which is decidable. We prove the (correctness of the) construction by structural induction on φ . In the base case φ is an atomic formula and we distinguish the following cases.

1. If $\varphi = (x \rightarrow y)$ then we use Theorem 4 to obtain an anchored GTT for \rightarrow_ε , which is transformed into an RR_2 automaton for $\langle \rightarrow_\varepsilon \rangle$ by Theorem 10. An application of Theorem 11 with $n = 1$ and $p = \geq$ yields an RR_2 automaton for $\langle (\rightarrow_\varepsilon) \supseteq^1 \rangle = \llbracket \varphi \rrbracket$.
2. If $\varphi = (x \rightarrow^* y)$ then we repeat the constructions in the previous case, with an additional application of modified transitive closure (Theorem 8) before Theorem 11 (with $n = p = \geq$) is applied.
3. If $\varphi = (x = y)$ then $\llbracket \varphi \rrbracket$ is regular by Lemma 18.

Here we assume that $x \neq y$. If x and y are the same variable then $\llbracket \varphi \rrbracket$ is a set of ground terms and the above constructions need to be modified as follows. If $\varphi = (x = x)$ then $\llbracket \varphi \rrbracket = \{\langle t \rangle \mid t \in \mathcal{T}(\mathcal{F})\} = \mathcal{T}(\mathcal{F})$ is accepted by the tree automaton $(\mathcal{F}, \{q\}, \{q\}, \Delta)$ with Δ consisting of all rules $f(q, \dots, q) \rightarrow q$ for $f \in \mathcal{F}$. Consider $\varphi = (x \rightarrow x)$. We have $\{\langle t, t \rangle \mid t \rightarrow_{\mathcal{R}} t\} = \{\langle t, u \rangle \mid t \rightarrow_{\mathcal{R}} u \text{ and } t = u\}$. The latter is regular (cases 1 and 3 above

together with Theorem 12) and hence the regularity of $\langle \llbracket \varphi \rrbracket \rangle = \{ \langle t \mid t \rightarrow_{\mathcal{R}} t \} \}$ follows by an application of Theorem 2. In the remaining case ($\varphi = (x \rightarrow^* x)$) we reason as in the previous case (using cases 2 and 3 above). Next we consider the propositional connectives.

4. Suppose $\varphi = \neg\psi$. The induction hypothesis yields an RR_n automaton that accepts $\langle \llbracket \psi \rrbracket \rangle$. Since the class of n -ary regular relations is effectively closed under complement (Theorem 13), we obtain an RR_n automaton that accepts $\langle \llbracket \varphi \rrbracket \rangle$.
5. Suppose $\varphi = \psi_1 \wedge \psi_2$. Since ψ_1 and ψ_2 may have less free variables than φ , we cannot use Theorem 12 without further ado. Let y_1, \dots, y_k be the free variables in ψ_1 and z_1, \dots, z_m be the free variables in ψ_2 . We have $\{x_1, \dots, x_n\} = \{y_1, \dots, y_k\} \cup \{z_1, \dots, z_m\}$. Because regular relations are closed under permutation (Theorem 14), we may assume that the variables in y_1, \dots, y_k and z_1, \dots, z_m are listed in the same order as in x_1, \dots, x_n . The induction hypothesis yields an RR_k automaton \mathcal{A}_1 for $\langle \llbracket \psi_1 \rrbracket \rangle$ and an RR_m automaton \mathcal{A}_2 for $\langle \llbracket \psi_2 \rrbracket \rangle$. Using $2n - (k + m)$ applications of cylindrification (Theorem 14), these automata are turned into RR_n automata. Since n -ary regular relations are closed under intersection (Theorem 12), we obtain an RR_n automaton for $\langle \llbracket \varphi \rrbracket \rangle$.
6. The other binary connectives are handled exactly like conjunction.

The final cases involve the two quantifiers.

7. Suppose $\varphi = \exists x \psi$. If x does not occur free in ψ then $\langle \llbracket \varphi \rrbracket \rangle = \langle \llbracket \psi \rrbracket \rangle$ and hence the result follows immediately from the induction hypothesis. So we assume that x occurs free in ψ and $n \geq 0$. The induction hypothesis yields an RR_{n+1} automaton that accepts $\langle \llbracket \psi \rrbracket \rangle$. Since the class of regular relations is effectively closed under projection (Theorem 2), we obtain an RR_n automaton that accepts $\langle \llbracket \varphi \rrbracket \rangle$.
8. The case $\varphi = \forall x \psi$ reduces to the preceding case by the well-known equivalence $\forall x \psi \equiv \neg \exists x \neg \psi$. □

6 Properties on Non-ground Terms

Since tree automata operate on ground terms, the decision procedure presented in the preceding section is restricted to properties on ground terms. The following example shows that ground-confluence, i.e., confluence restricted to ground terms, is not the same as confluence.

Example 22 The left-linear right-ground TRS \mathcal{R} consisting of the rules

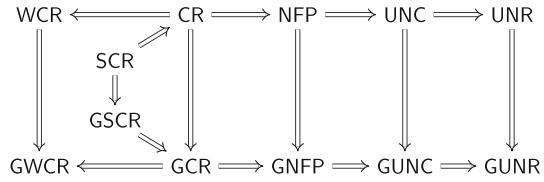
$$a \rightarrow b \qquad f(a, x) \rightarrow b \qquad f(b, b) \rightarrow b$$

over the signature $\mathcal{F} = \{a, b, f\}$ is ground-confluent because every ground term in $\mathcal{T}(\mathcal{F})$ rewrites to b . Confluence does not hold; the term $f(a, x)$ rewrites to the different normal forms b and $f(b, x)$.

In this section we present results that allow the use of FORT on (certain) properties over arbitrary terms. The main idea is to extend the given signature \mathcal{F} with constants to replace variables in terms. The required number of additional constants depends on the property under consideration. We consider the following confluence-related properties:

CR: $\forall s \forall t \forall u (s \rightarrow^* t \wedge s \rightarrow^* u \implies t \downarrow u)$	confluence
SCR: $\forall s \forall t \forall u (s \rightarrow t \wedge s \rightarrow u \implies \exists v (t \rightarrow^= v \wedge u \rightarrow^* v))$	strong confluence
WCR: $\forall s \forall t \forall u (s \rightarrow t \wedge s \rightarrow u \implies t \downarrow u)$	local confluence

Fig. 6 Confluence-related properties on ground and non-ground terms



NFP: $\forall s \forall t \forall u (s \rightarrow^* t \wedge s \rightarrow^! u \implies t \rightarrow^! u)$ normal form property

UNR: $\forall s \forall t \forall u (s \rightarrow^! t \wedge s \rightarrow^! u \implies t = u)$
 unique normal forms with respect to reduction

UNC: $\forall t \forall u (t \leftrightarrow^* u \wedge \text{NF}(t) \wedge \text{NF}(u) \implies t = u)$
 unique normal forms with respect to conversion

Here $t \downarrow u$ denotes joinability: $\exists v (t \rightarrow^* v \wedge u \rightarrow^* v)$. Let \mathcal{P}_1 be the collection of these properties. We also consider the following properties involving two TRSs \mathcal{R} and \mathcal{S} :

COM: $\forall s \forall t \forall u (s \rightarrow_{\mathcal{R}}^* t \wedge s \rightarrow_{\mathcal{S}}^* u \implies \exists v (t \rightarrow_{\mathcal{S}}^* v \wedge u \rightarrow_{\mathcal{R}}^* v))$ commutation

CE: $\forall s \forall t (s \leftrightarrow_{\mathcal{R}}^* t \iff s \leftrightarrow_{\mathcal{S}}^* t)$ conversion equivalence

NE: $\forall s \forall t (s \rightarrow_{\mathcal{R}}^! t \iff s \rightarrow_{\mathcal{S}}^! t)$ normalization equivalence

Let $\mathcal{P}_2 = \{\text{COM}, \text{CE}, \text{NE}\}$. For a property $P \in \mathcal{P}_1 \cup \mathcal{P}_2$, GP denotes the property P restricted to ground terms. The diagram in Fig. 6 summarizes the relationships between properties P and GP for $P \in \mathcal{P}_1$. The properties $\text{CE}, \text{NE} \in \mathcal{P}_2$ are unrelated.

According to the following result, all considered properties are closed under signature extension.

Lemma 27 *Let \mathcal{R} and \mathcal{S} be linear variable-separated TRSs over a common signature \mathcal{F} .*

1. *If $P \in \mathcal{P}_1$ and $(\mathcal{F}, \mathcal{R}) \models P$ then $(\mathcal{F} \uplus \{c\}, \mathcal{R}) \models P$.*
2. *If $P \in \mathcal{P}_2$ and $(\mathcal{F}, \mathcal{R}, \mathcal{S}) \models P$ then $(\mathcal{F} \uplus \{c\}, \mathcal{R}, \mathcal{S}) \models P$.*

Proof Let \mathcal{U} be a linear variable-separated TRS not containing the constant c . For any $x \in \mathcal{V}$, the mapping $\phi_c^x: \mathcal{T}(\mathcal{F} \uplus \{c\}, \mathcal{V}) \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{V})$ replaces all occurrences of c in terms by the variable x :

$$\phi_c^x(t) = \begin{cases} x & \text{if } t = c \\ t & \text{if } t \in \mathcal{V} \\ f(\phi_c^x(t_1), \dots, \phi_c^x(t_n)) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

A straightforward induction proof reveals that $\phi_c^x(s) \rightarrow_{\mathcal{U}}^* \phi_c^x(t)$ whenever $s \rightarrow_{\mathcal{U}}^* t$. By choosing $x \notin \text{Var}(s) \cup \text{Var}(t)$, the reverse direction holds as well. Moreover, since linear variable-separated TRSs are closed under rule inversion, the equivalence also holds for $\leftrightarrow_{\mathcal{U}}^* = \rightarrow_{\mathcal{U} \cup \mathcal{U}^{-1}}^*$. The lemma is an easy consequence of these facts. We illustrate this for COM. Given $s \rightarrow_{\mathcal{R}}^* t$ and $s \rightarrow_{\mathcal{S}}^* u$, with $s, t, u \in \mathcal{T}(\mathcal{F} \uplus \{c\}, \mathcal{V})$, we obtain $\phi_c^x(s) \rightarrow_{\mathcal{R}}^* \phi_c^x(t)$ and $\phi_c^x(s) \rightarrow_{\mathcal{S}}^* \phi_c^x(u)$. Commutation of $(\mathcal{F}, \mathcal{R}, \mathcal{S})$ yields a term $v \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $\phi_c^x(t) \rightarrow_{\mathcal{S}}^* v$ and $\phi_c^x(u) \rightarrow_{\mathcal{R}}^* v$. By taking $x \notin \text{Var}(t) \cup \text{Var}(u)$, we obtain $t \rightarrow_{\mathcal{S}}^* v'$ and $u \rightarrow_{\mathcal{R}}^* v'$ for $v' = v\{x \mapsto c\}$ by closure of rewriting under substitutions. ✔✔✔

So adding constants preserves the properties of interest. For removing constants more effort is required. For the properties in $\mathcal{P}_1 \cup \mathcal{P}_2$, root steps will play a major role. Root

steps are important since they permit the use of different substitutions for the left and right-hand side of the employed rewrite rule, due to variable separation. We therefore start with a preliminary result (Lemma 28) which provides abstract conditions that permit the restriction to rewrite sequences containing root steps. We write $\rightarrow_{\mathcal{R}}^{*\varepsilon*}$ for the relation $\rightarrow_{\mathcal{R}}^* \cdot \rightarrow_{\mathcal{R}}^{\varepsilon} \cdot \rightarrow_{\mathcal{R}}^*$. The proof of Lemma 28 is obtained by a straightforward induction on the term structure and the multi-hole context closure of the rewrite relation, and is omitted.

Definition 18 A binary predicate P on terms over a given signature \mathcal{F} is closed under *multi-hole contexts* if $P(C[s_1, \dots, s_n], C[t_1, \dots, t_n])$ holds whenever C is a multi-hole context over \mathcal{F} with $n \geq 0$ holes and $P(s_i, t_i)$ holds for all $1 \leq i \leq n$. ✔

Lemma 28 Let \mathcal{A} and \mathcal{B} be TRSs over the same signature \mathcal{F} and let P be a binary predicate that is closed under multi-hole contexts over \mathcal{F} .

1. If $s \rightarrow_{\mathcal{A}}^{*\varepsilon*} t \implies P(s, t)$ for all terms s and t then $s \rightarrow_{\mathcal{A}}^* t \implies P(s, t)$ for all terms s and t .
2. If $s \rightarrow_{\mathcal{A}}^{*\varepsilon*} \cdot \rightarrow_{\mathcal{B}}^* t \vee s \rightarrow_{\mathcal{A}}^* \cdot \rightarrow_{\mathcal{B}}^{*\varepsilon*} t \implies P(s, t)$ for all terms s and t then $s \rightarrow_{\mathcal{A}}^* \cdot \rightarrow_{\mathcal{B}}^* t \implies P(s, t)$ for all terms s and t . ✔✔

For example, in the results below (Lemmata 34 and 35) for NFP we make use of this lemma by instantiating part 2 with $P_1(s, t) : \text{NF}(t) \implies s \rightarrow_{\mathcal{R}}^* t, \mathcal{R}^-$ for \mathcal{A} , and \mathcal{R} for \mathcal{B} . This results in the statement that if

$$s \rightarrow_{\mathcal{R}^-}^{*\varepsilon*} \cdot \rightarrow_{\mathcal{R}}^* t \vee s \rightarrow_{\mathcal{R}^-}^* \cdot \rightarrow_{\mathcal{R}}^{*\varepsilon*} t \implies \text{NF}(t) \implies s \rightarrow_{\mathcal{R}}^* t$$

then

$$s \rightarrow_{\mathcal{R}^-}^* \cdot \rightarrow_{\mathcal{R}}^* t \implies \text{NF}(t) \implies s \rightarrow_{\mathcal{R}}^* t$$

Using the identity $\rightarrow_{\mathcal{R}^-} = \mathcal{R} \leftarrow$ and the definition of NFP, it follows that NFP is a consequence of the statement

$$s \xrightarrow{\mathcal{R} \leftarrow}^{*\varepsilon*} \cdot \rightarrow_{\mathcal{R}}^* t \vee s \xrightarrow{\mathcal{R} \leftarrow}^* \cdot \rightarrow_{\mathcal{R}}^{*\varepsilon*} t \implies \text{NF}(t) \implies s \rightarrow_{\mathcal{R}}^* t$$

for all $s, t \in \mathcal{T}(\mathcal{F})$. Hence we only need to consider rewrite sequences involving root steps, which together with variable separation significantly simplifies the proof. For the other properties of interest, Lemma 28 is instantiated as follows.

- For UNC we use part 1 with $P_2(s, t) : \text{NF}(s) \wedge \text{NF}(t) \implies s = t$ and $\mathcal{R} \cup \mathcal{R}^-$ for \mathcal{A} .
- For UNR we use part 2 with the same predicate P_2 and \mathcal{R}^- for \mathcal{A} and \mathcal{R} for \mathcal{B} .
- For COM we use part 2 with $P_3(s, t) : s \rightarrow_{\mathcal{S}}^* \cdot \rightarrow_{\mathcal{R}^-}^* t$ and \mathcal{R}^- for \mathcal{A} and \mathcal{S} for \mathcal{B} .
- For CR we use part 2 with the same predicate P_3 and replace \mathcal{S} by \mathcal{R} .
- For NE we use part 1 twice, with $P_4(s, t) : \text{NF}_{\mathcal{R}}(t) \implies s \rightarrow_{\mathcal{S}}^* t$ and \mathcal{R} for \mathcal{A} , and with $P_5(s, t) : \text{NF}_{\mathcal{S}}(t) \implies s \rightarrow_{\mathcal{R}}^* t$ and \mathcal{S} for \mathcal{A} .
- For CE we use part 1 twice, with $P_6(s, t) : s \rightarrow_{\mathcal{S} \cup \mathcal{S}^-}^* t$ and $\mathcal{R} \cup \mathcal{R}^-$ for \mathcal{A} , and with $P_7(s, t) : s \rightarrow_{\mathcal{R} \cup \mathcal{R}^-}^* t$ and $\mathcal{S} \cup \mathcal{S}^-$ for \mathcal{A} .

In addition, we make use of the identities $\rightarrow_{\mathcal{R} \cup \mathcal{R}^-}^{*\varepsilon*} = \leftrightarrow_{\mathcal{R}}^{*\varepsilon*}$ and $\rightarrow_{\mathcal{R} \cup \mathcal{R}^-}^* = \leftrightarrow_{\mathcal{R}}^*$ for UNC and CE.

Lemma 29 The properties P_1, \dots, P_7 are closed under multi-hole contexts. ✔✔✔✔

Strong confluence (SCR) and local confluence (WCR) cannot be reduced to root steps with Lemma 28, because they involve single steps in their definition, which are not multi-hole context closed. However, by investigating the positions involved in $s \rightarrow t$ and $s \rightarrow u$ we easily deduce a reduction to root steps for both properties.

Lemma 30 *A TRS is local confluent if and only if*

$$s \rightarrow^\varepsilon t \wedge s \rightarrow u \implies t \downarrow u$$

for all terms s, t and u . A TRS is strongly confluent if and only if

$$s \rightarrow^\varepsilon t \wedge s \rightarrow u \vee s \rightarrow t \wedge s \rightarrow^\varepsilon u \implies t \rightarrow^= \cdot \leftarrow^* u$$

for all terms s, t and u . ✔✔

The next lemma is a key result. It allows the removal of introduced fresh constants while preserving the reachability relation. Note that variable-separation is not required.

Lemma 31 *Let \mathcal{R} be a linear TRS over a signature \mathcal{F} that contains a constant c which does not appear in \mathcal{R} . If $s \rightarrow_{\mathcal{R}}^* t$ with $c \in \mathcal{F}\text{un}(s) \setminus \mathcal{F}\text{un}(t)$ then $s[u]_p \rightarrow_{\mathcal{R}}^* t$ using the same rewrite rules at the same positions, for all terms u and positions $p \in \mathcal{P}\text{os}(s)$ such that $s|_p = c$. ✔✔*

The restriction to linear TRSs can also be lifted, at the expense of a more complicated replacement function and proof. Since the decision procedure implemented in FORT-h relies on linearity and variable-separation, we present a simple proof for linear TRSs. Due to calculations involving positions, the formalization in Isabelle/HOL was anything but simple.

Proof We use induction on the length of $s \rightarrow_{\mathcal{R}}^* t$. If this length is zero then there is nothing to show as $\mathcal{F}\text{un}(s) \setminus \mathcal{F}\text{un}(t) = \emptyset$. Suppose $s \rightarrow_{\mathcal{R}} v \rightarrow_{\mathcal{R}}^* t$ and write $s = C[\ell\sigma] \rightarrow_{\mathcal{R}} C[r\sigma] = v$. Let p' be the position of the hole in C and let $p \in \mathcal{P}\text{os}(s)$ with $s|_p = c$. We distinguish two cases.

If $p' \parallel p$ then $s[u]_p = (C[u]_{p'})[\ell\sigma]_{p'} \rightarrow_{\mathcal{R}} v'$ with $v' = (C[u]_{p'})[r\sigma]_{p'}$. Since $v|_p = C|_p = c$ we can apply the induction hypothesis to $v \rightarrow_{\mathcal{R}}^* t$. This yields $v' \rightarrow_{\mathcal{R}}^* t$ and hence $s[u]_p \rightarrow_{\mathcal{R}}^* t$ as desired.

In the remaining case, $p' \leq p$. From $s|_p = c$ and the fact that c does not appear in \mathcal{R} we infer that there exists a variable $y \in \mathcal{V}\text{ar}(\ell)$ such that $c \in \mathcal{F}\text{un}(\sigma(y))$. Let q be the (unique) position of y in ℓ and consider the substitution

$$\tau(x) = \begin{cases} \sigma(y)[u]_{q'} & \text{if } x = y \\ \sigma(x) & \text{otherwise} \end{cases}$$

Here $q' = p \setminus (p'q)$ is the position of c in $\sigma(y)$. If $y \notin \mathcal{V}\text{ar}(r)$ then $v = C[r\sigma] = C[r\tau]$ and thus $s[u]_p = C[\ell\tau] \rightarrow_{\mathcal{R}} C[r\tau] = v \rightarrow_{\mathcal{R}}^* t$. If $y \in \mathcal{V}\text{ar}(r)$ then there exists a unique position $q'' \in \mathcal{P}\text{os}(r)$ such that $r|_{q''} = y$. So $v|_{p'q''q'} = c$ and we obtain $s[u]_p = C[\ell\tau] \rightarrow_{\mathcal{R}} C[r\tau] = v[u]_{p'q''q'} \rightarrow_{\mathcal{R}}^* t$ from the induction hypothesis. □

In the proofs below Lemma 31 (also for \mathcal{R}^-) is used as follows. Let σ_c denote the substitution mapping all variables to c . If $s\sigma_c \rightarrow_{\mathcal{R}}^* t$ then $s \rightarrow_{\mathcal{R}}^* t$ by repeated applications of Lemma 31 (if the conditions are satisfied).

We now prove that two fresh constants are sufficient to reduce commutation (COM), confluence (CR), local confluence (WCR), unique normal forms (UNC and UNR), and the normal form property (NFP) to the corresponding ground properties.


Lemma 32 *Linear variable-separated TRSs \mathcal{R} and S over a common signature \mathcal{F} commute if and only if \mathcal{R} and S ground-commute over $\mathcal{F} \uplus \{c, d\}$. ✔*

Proof The only-if direction follows from Lemma 27. For the if direction suppose \mathcal{R} and \mathcal{S} ground-commute on terms in $\mathcal{T}(\mathcal{F} \uplus \{c, d\})$. In order to conclude that \mathcal{R} and \mathcal{S} commute on terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$, according to Lemma 28, it suffices to show the inclusions

$$\rightarrow_{\mathcal{R}^-}^{*\varepsilon*} \cdot \rightarrow_{\mathcal{S}}^* \subseteq \rightarrow_{\mathcal{S}}^* \cdot \rightarrow_{\mathcal{R}^-}^* \qquad \rightarrow_{\mathcal{R}^-}^* \cdot \rightarrow_{\mathcal{S}}^{*\varepsilon*} \subseteq \rightarrow_{\mathcal{S}}^* \cdot \rightarrow_{\mathcal{R}^-}^*$$

on terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Suppose $s \rightarrow_{\mathcal{R}^-}^{*\varepsilon*} \cdot \rightarrow_{\mathcal{S}}^* t$. Let the substitution σ_c map all variables to c and let σ_d map all variables to d . Since rewriting is closed under substitutions and the variable-separated rule used in the root step $\rightarrow_{\mathcal{R}^-}^\varepsilon$ allows changing the substitution, we obtain $s\sigma_c \rightarrow_{\mathcal{R}^-}^{*\varepsilon*} \cdot \rightarrow_{\mathcal{S}}^* t\sigma_d$. From ground commutation we obtain $s\sigma_c \rightarrow_{\mathcal{S}}^* \cdot \rightarrow_{\mathcal{R}^-}^* t\sigma_d$. Note that s and t are terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and hence do not contain the constants c and d . Therefore, $d \notin \text{Fun}(s\sigma_c)$ and $c \notin \text{Fun}(t\sigma_d)$. As a consequence, repeated applications of Lemma 31 transform $s\sigma_c \rightarrow_{\mathcal{S}}^* \cdot \rightarrow_{\mathcal{R}^-}^* t\sigma_d$ into a sequence $s \rightarrow_{\mathcal{S}}^* \cdot \rightarrow_{\mathcal{R}^-}^* t$ in which c and d do not appear, proving the first inclusion. Note that in our setting TRSs are closed under rule reversal. Hence we can apply Lemma 31 in both directions, which allows us to remove the constant d from the term t . The second inclusion $\rightarrow_{\mathcal{R}^-}^* \cdot \rightarrow_{\mathcal{S}}^{*\varepsilon*} \subseteq \rightarrow_{\mathcal{S}}^* \cdot \rightarrow_{\mathcal{R}^-}^*$ is obtained in the same way. \square

If the TRSs \mathcal{R} and \mathcal{S} are left-linear right-ground (as opposed to linear variable-separated) then the term t in the above proof is ground due to the root step involved. Hence $t\sigma_d = t$, which allows us to simplify the proof and strengthen the statement to use only one additional constant.

Lemma 33 *Left-linear right-ground TRSs \mathcal{R} and \mathcal{S} over a common signature \mathcal{F} commute if and only if \mathcal{R} and \mathcal{S} ground-commute over $\mathcal{F} \uplus \{c\}$.* 

The proof for confluence follows directly from commutation. The proofs for the other properties in \mathcal{P}_1 are obtained in a similar manner. We present the proof details for strong confluence since it requires a bit more effort.

Lemma 34 *Let \mathcal{R} be a linear variable-separated TRS over a signature \mathcal{F} . If $P \in \mathcal{P}_1$ then*

$$(\mathcal{F}, \mathcal{R}) \models P \iff (\mathcal{F} \uplus \{c, d\}, \mathcal{R}) \models GP \qquad \img alt="five checkmark icons" data-bbox="738 585 890 600"/>$$

Proof We present the if direction for $P = \text{SCR}$. First we use Lemma 30 to reduce the problem to local peaks involving a root step. Following the reasoning in the proof of Lemma 32, we obtain a witness v such that $t\sigma_d \rightarrow_{\mathcal{R}}^= v \xrightarrow{\mathcal{R}^-}^* u\sigma_c$. If $t\sigma_d = v$ then $u\sigma_c \rightarrow_{\mathcal{R}}^* t\sigma_d$ and we obtain $u \rightarrow_{\mathcal{R}}^* t$ with the help of Lemma 31. So assume $u\sigma_c \rightarrow_{\mathcal{R}}^* \cdot \rightarrow_{\mathcal{R}^-}^* t\sigma_d$. Using Lemma 31 and induction on the number of variables in u we deduce $u \rightarrow_{\mathcal{R}}^* \cdot \rightarrow_{\mathcal{R}^-}^* t\sigma_d$. The same argument applied to t produces $u \rightarrow_{\mathcal{R}}^* w \rightarrow_{\mathcal{R}^-}^* t$. Note that w may contain occurrences of the constants c and d since \mathcal{R} is a variable-separated TRS. We use the map defined in the proof of Lemma 27 to eliminate these: $u = \phi_c^x(\phi_d^x(u)) \rightarrow_{\mathcal{R}}^* \phi_c^x(\phi_d^x(w)) \rightarrow_{\mathcal{R}^-}^* \phi_c^x(\phi_d^x(t)) = t$. \square

Lemma 35 *Let \mathcal{R} be a left-linear right-ground TRS over a signature \mathcal{F} . If $P \in \mathcal{P}_1 \setminus \{\text{UNC}\}$ then*

$$(\mathcal{F}, \mathcal{R}) \models P \iff (\mathcal{F} \uplus \{c\}, \mathcal{R}) \models GP$$

Moreover,

$$(\mathcal{F}, \mathcal{R}) \models \text{UNC} \iff (\mathcal{F} \uplus \{c, d\}, \mathcal{R}) \models \text{GUNC} \qquad \img alt="five checkmark icons" data-bbox="765 885 890 900"/>$$

The simplification in the proof of Lemma 32 for left-linear right-ground systems is not applicable for UNC as conversion can introduce variables. The following example shows that adding a single fresh constant is indeed insufficient for UNC.

Example 23 The left-linear right-ground TRS \mathcal{R} consisting of the rules

$$a \rightarrow b \quad f(x, a) \rightarrow f(b, b) \quad f(b, x) \rightarrow f(b, b) \quad f(f(x, y), z) \rightarrow f(b, b)$$

does not satisfy UNC since $f(x, b) \leftarrow f(x, a) \rightarrow f(b, b) \leftarrow f(y, a) \rightarrow f(y, b)$ is a conversion between distinct normal forms. Adding a single fresh constant c is not enough to violate GUNC as the last two rewrite rules ensure that $f(c, b)$ is the only ground instance of $f(x, b)$ that is a normal form. Adding another fresh constant d , GUNC is lost: $f(c, b) \leftarrow f(c, a) \rightarrow f(b, b) \leftarrow f(d, a) \rightarrow f(d, b)$.

The following example shows that at least two fresh constants are required to reduce confluence to ground-confluence for linear variable-separated TRSs.

Example 24 Consider the linear variable-separated TRS \mathcal{R} consisting of the single rule $a \rightarrow x$ over the signature $\mathcal{F} = \{a\}$. Since $x \mathcal{R} \leftarrow a \rightarrow_{\mathcal{R}} y$ with distinct variables x and y , \mathcal{R} is not confluent. Ground-confluence holds trivially as $a \rightarrow_{\mathcal{R}} a$ is the only rewrite step between ground terms. Adding a single fresh constant b does not destroy ground-confluence ($a \rightarrow_{\mathcal{R}} a$ and $a \rightarrow_{\mathcal{R}} b$ are the only steps). By adding a second fresh constant c , ground-confluence is lost: $b \mathcal{R} \leftarrow a \rightarrow_{\mathcal{R}} c$.

We now turn our attention to the equivalence properties (CE and NE) in \mathcal{P}_2 . For conversion equivalence a single fresh constant suffices to reduce it to ground conversion equivalence.

Lemma 36 *Linear variable-separated TRSs \mathcal{R} and S over a common signature \mathcal{F} such that $\mathcal{T}(\mathcal{F}) \neq \emptyset$ are conversion equivalent if and only if \mathcal{R} and S are ground conversion equivalent over $\mathcal{F} \uplus \{c\}$.* ✔

Proof For the if direction we assume that \mathcal{R} and S are ground conversion equivalent over $\mathcal{F} \uplus \{c\}$. Due Lemma 28 and symmetry, it suffices to show the inclusion $\leftrightarrow_{\mathcal{R}}^{*\varepsilon*} \subseteq \leftrightarrow_S^*$ on terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Suppose $s \leftrightarrow_{\mathcal{R}}^{*\varepsilon*} t$. Let $d \in \mathcal{F}$ be a constant, whose existence is guaranteed by the assumption $\mathcal{T}(\mathcal{F}) \neq \emptyset$, and consider the substitutions σ_c and σ_d in the proof of Lemma 32. Closure under substitutions and variable separation yields $s\sigma_c \leftrightarrow_{\mathcal{R}}^{*\varepsilon*} t\sigma_c$ and $s\sigma_c \leftrightarrow_{\mathcal{R}}^{*\varepsilon*} t\sigma_d$. Ground conversion equivalence gives $s\sigma_c \leftrightarrow_S^* t\sigma_c$ and $s\sigma_c \leftrightarrow_S^* t\sigma_d$, and thus also $t\sigma_c \leftrightarrow_S^* t\sigma_d$. Using Lemma 31 yields $s \leftrightarrow_S^* t\sigma_d$ and $t \leftrightarrow_S^* t\sigma_d$. Hence $s \leftrightarrow_S^* t$ as desired. The only-if direction easily follows from Lemma 27. □

Two fresh constants are required to reduce normalization equivalence to its ground version.

Lemma 37 *Linear variable-separated TRSs \mathcal{R} and S over a common signature \mathcal{F} are normalization equivalent if and only if \mathcal{R} and S are ground normalization equivalent over $\mathcal{F} \uplus \{c, d\}$.* ✔

Proof For the if direction we assume that \mathcal{R} and S are ground normalization equivalent over $\mathcal{F} \uplus \{c, d\}$. Note that this implies that $\text{NF}_{\mathcal{R}}(t) \iff \text{NF}_S(t)$ for all terms t . We apply Lemma 28 and symmetry, reducing the problem to $s \rightarrow_{\mathcal{R}}^{*\varepsilon*} t \implies \text{NF}_{\mathcal{R}}(t) \implies s \rightarrow_S^* t$. Let σ_c and σ_d be substitutions replacing all variables by c and d respectively. Closure under substitution and variable separation yields $s\sigma_c \rightarrow_{\mathcal{R}}^{*\varepsilon*} t\sigma_d$, and $\text{NF}_{\mathcal{R}}(t\sigma_d)$ since d does not appear in \mathcal{R} . Ground normalization equivalence gives $s\sigma_c \rightarrow_S^* t\sigma_d$. Applying Lemma 31 we obtain the desired $s \rightarrow_S^* t$. The only-if direction follows from Lemma 27. □

Contrary to Lemma 36 one fresh constant is not sufficient as seen by the following counterexample.

Example 25 Consider the two linear variable-separated TRSs

$$\begin{array}{lll}
 \mathcal{R}: & a \rightarrow b & f(f(x, y), z) \rightarrow f(b, b) & f(b, x) \rightarrow f(b, b) \\
 & f(x, a) \rightarrow f(z, b) & & \\
 \mathcal{S}: & a \rightarrow b & f(f(x, y), z) \rightarrow f(b, b) & f(b, x) \rightarrow f(b, b) \\
 & f(b, a) \rightarrow f(z, b) & f(f(x, y), a) \rightarrow f(z, b) &
 \end{array}$$

They are not normalization equivalent since $f(x, a) \xrightarrow{!}_{\mathcal{R}} f(z, b)$ and $f(x, a) \not\xrightarrow{!}_{\mathcal{S}} *f(z, b)$. The TRSs are however ground normalization equivalent over the signature $\mathcal{F} \uplus \{c\}$. First observe that the only ground normal forms reachable via a rewrite sequence involving a root step are b and $f(c, b)$. The normal form b is reached (using a root step) only from a , in both \mathcal{R} and \mathcal{S} . The normal form $f(c, b)$ can be reached from all ground terms of the shape $f(t, a)$. For \mathcal{R} this is obvious and for \mathcal{S} this can be seen by a case analysis on the root symbol of t . Adding a second constant d allows one to mimick the original counterexample since $f(c, a) \xrightarrow{!}_{\mathcal{R}} f(d, b)$ and $f(c, a) \not\xrightarrow{!}_{\mathcal{S}} *f(d, b)$.

For left-linear right-ground TRSs, a single fresh constant is enough to reduce normalization equivalence to ground normalization equivalence.

Lemma 38 *Left-linear right-ground TRSs \mathcal{R} and \mathcal{S} over a common signature \mathcal{F} are normalization equivalent if and only if \mathcal{R} and \mathcal{S} are ground normalization equivalent over $\mathcal{F} \uplus \{c\}$.* ☑

Proof We mention the differences with the proof of Lemma 37. For the equivalence of $NF_{\mathcal{R}}(t)$ and $NF_{\mathcal{S}}(t)$ for arbitrary terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, a single constant suffices. If $s \xrightarrow{*}_{\mathcal{R}} t$ then t is ground. Hence $s\sigma_c \xrightarrow{*}_{\mathcal{R}} t$ and thus $s\sigma_c \xrightarrow{*}_{\mathcal{S}} t$ by ground normalization equivalence. Lemma 31 gives $s \xrightarrow{*}_{\mathcal{S}} t$. □

Each additional constant can increase the execution time of FORT-h significantly, as seen later in Example 36. Hence results that reduce the required number are of obvious interest. In the remainder of this section we present results for ground TRSs and for TRSs over *monadic* signatures, which are signatures that consist of constants and unary function symbols.

Lemma 39 *Let \mathcal{R} and \mathcal{S} be right-ground TRSs over a signature \mathcal{F} . If \mathcal{R} and \mathcal{S} are ground or \mathcal{F} is monadic then*

$$\begin{aligned}
 (\mathcal{F}, \mathcal{R}) \models P & \iff (\mathcal{F}, \mathcal{R}) \models GP \text{ for all } P \in \mathcal{P}_1 && \text{☑☑☑☑☑☑☑☑☑☑} \\
 (\mathcal{F}, \mathcal{R}, \mathcal{S}) \models COM & \iff (\mathcal{F}, \mathcal{R}, \mathcal{S}) \models GCOM && \text{☑☑}
 \end{aligned}$$

Proof First assume that \mathcal{R} is ground. In this case only ground subterms can be rewritten. Given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we write $t = C[[t_1, \dots, t_n]]$ if $t = C[t_1, \dots, t_n]$ and t_1, \dots, t_n are the maximal ground subterms of t . So all variables appearing in t occur in C . The following property is obvious:

- (a) if $t = C[[t_1, \dots, t_n]] \xrightarrow{*}_{\mathcal{R}} u$ then $u = C[[u_1, \dots, u_n]]$ and $t_i \xrightarrow{*}_{\mathcal{R}} u_i$ for all $1 \leq i \leq n$.

Suppose $(\mathcal{F}, \mathcal{R}) \models \text{GCR}$ and consider $s \rightarrow_{\mathcal{R}}^* t$ and $s \rightarrow_{\mathcal{R}}^* u$ with $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. Writing $s = C[[s_1, \dots, s_n]]$, we obtain $t = C[[t_1, \dots, t_n]]$ and $u = C[[u_1, \dots, u_n]]$ with $s_i \rightarrow_{\mathcal{R}}^* t_i$ and $s_i \rightarrow_{\mathcal{R}}^* u_i$ for all $1 \leq i \leq n$. GCR yields $t_i \downarrow u_i$ for all $1 \leq i \leq n$. Hence $t \downarrow u$ as desired. The proofs for the other properties in \mathcal{P} are equally easy. For UNC we note that $\leftrightarrow_{\mathcal{R}}^*$ coincides with $\rightarrow_{\mathcal{R} \cup \mathcal{R}^-}^*$ for the *ground* TRS $\mathcal{R} \cup \mathcal{R}^-$.

Next suppose that \mathcal{F} is monadic. Let $(\mathcal{F}, \mathcal{R}) \models \text{GP}$ and let σ be the substitution that maps all variables to some arbitrary but fixed ground term. In this case the following property holds:

(b) if $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $t \rightarrow u$ then $u \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $t\sigma \rightarrow u$.

We consider $P = \text{NFP}$ and $P = \text{UNC}$ and leave the proof for the other properties to the reader. Let $s \rightarrow_{\mathcal{R}} t$ and $s \rightarrow_{\mathcal{R}}^! u$. We obtain $s\sigma \rightarrow_{\mathcal{R}} t$ and $s\sigma \rightarrow_{\mathcal{R}}^! u$ from property 2. (Note that $s \neq u$.) Hence $t \rightarrow_{\mathcal{R}}^* u$ follows from GNFP. Let $t \leftrightarrow_{\mathcal{R}}^* u$ with normal forms t and u . If t and u are ground terms then we obtain $t = u$ from GUNC (after applying the substitution σ to all intermediate terms in the conversion between t and u). Otherwise, the conversion between t and u must be empty due to property (b) and the fact that t and u are normal forms. Hence also in this case $t = u$. □

In contrast to COM, the properties NE and CE require additional constants for TRSs over monadic signatures.

Example 26 The linear variable-separated TRSs

$$\mathcal{R}: f(x) \rightarrow a \qquad \mathcal{S}: f(a) \rightarrow a \quad f(f(a)) \rightarrow a$$

are neither normalization equivalent nor conversion equivalent as can be seen from $f(x) \rightarrow_{\mathcal{R}}^! a$ and $f(x) \not\leftrightarrow_{\mathcal{S}}^* a$. Since every ground term rewrites in \mathcal{R} and in \mathcal{S} to the unique ground normal form a , the TRSs are ground normalization equivalent as well as ground conversion equivalent.

Nevertheless, we can reduce the number of constants to one if the signature is monadic. A key observation is that in non-empty rewrite sequences in a linear variable-separated TRS over a monadic signature fresh constants can be replaced by arbitrary terms.

Lemma 40 *Let \mathcal{R} be a linear variable-separated TRS over a monadic signature \mathcal{F} that contains a constant c which does not appear in \mathcal{R} . If $s \rightarrow_{\mathcal{R}}^+ t$ and $p \in \text{Pos}(s)$ such that $s|_p = c$ then $s[u]_p \rightarrow_{\mathcal{R}}^+ t$ using the same rewrite rules at the same positions, for all terms u .* ✓

The proof follows the same structure as Lemma 31 and the details are left for the reader. As linear variable-separated TRSs are closed under inverse we can immediately deduce that rewrite sequences of the shape $s\sigma_c \rightarrow_{\mathcal{R}}^+ t\sigma_c$ imply $s \rightarrow_{\mathcal{R}}^+ t$ for monadic systems. With this we are ready to prove our claim.

Lemma 41 *Linear variable-separated TRSs \mathcal{R} and \mathcal{S} over a common monadic signature \mathcal{F} are normalization equivalent if and only if \mathcal{R} and \mathcal{S} are ground normalization equivalent over $\mathcal{F} \uplus \{c\}$.* ✓

Proof We again mention the differences with the proof of Lemma 37. For the equivalence of $\text{NF}_{\mathcal{R}}(t)$ and $\text{NF}_{\mathcal{S}}(t)$ for arbitrary terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, a single constant suffices. Consider a rewrite sequence $s \rightarrow_{\mathcal{R}}^{*\varepsilon*} t$ with $\text{NF}_{\mathcal{R}}(t)$. Ground normalization equivalence and substitution

Table 3 Additional constants required to reduce a property to the corresponding ground property

Property	Ground TRSs	Left-linear right-ground TRSs	Linear variable-separated TRSs
CR	0	1 (0)	2 (2)
SCR	0	1 (0)	2 (2)
WCR	0	1 (0)	2 (2)
COM	0	1 (0)	2 (2)
UNR	0	1 (0)	2 (2)
UNC	0	2 (0)	2 (2)
NFP	0	1 (0)	2 (2)
CE	0	1 (1)	1 (1)
NE	0	1 (1)	2 (1)

closure yields $s\sigma_c \rightarrow_{\mathcal{S}}^* t\sigma_c$. Furthermore, since the sequence $s \rightarrow_{\mathcal{R}}^{*\varepsilon*} t$ is non-empty by definition we know that $\neg \text{NF}_{\mathcal{R}}(s\sigma_c)$, which in turn yields $\neg \text{NF}_{\mathcal{S}}(s\sigma_c)$. Together with $\text{NF}_{\mathcal{S}}(t\sigma_c)$ this means $s\sigma_c \neq t\sigma_c$, and we obtain $s\sigma_c \rightarrow_{\mathcal{S}}^+ t\sigma_c$. Applying Lemma 40 twice allows us to replace c in $s\sigma_c$ and $t\sigma_c$ by the corresponding variables, leading to $s \rightarrow_{\mathcal{S}}^* t$. \square

The following example shows that we cannot reduce the number of constants (in Lemmata 32 and 34) for linear variable-separated TRSs over a monadic signature and properties $P \in \mathcal{P}_1 \cup \{\text{COM}\}$.

Example 27 The monadic linear variable-separated TRS \mathcal{R} consisting of the rules

$$\mathbf{g}(a) \rightarrow \mathbf{g}(x) \qquad \mathbf{g}(\mathbf{g}(x)) \rightarrow \mathbf{g}(y)$$

does not satisfy WCR and UNR, and hence also not CR, SCR, NFP and UNC, because $\mathbf{g}(x) \leftarrow \mathbf{g}(a) \rightarrow \mathbf{g}(y)$ with different normal forms $\mathbf{g}(x)$ and $\mathbf{g}(y)$. Adding a single fresh constant c is insufficient to violate GSCR and thus also GCR, GWCR, GNFP, GUNC and GUNR, because every term in $\mathcal{T}(\{\mathbf{g}, a, c\})$ can reach precisely one of the three ground normal forms a, c or $\mathbf{g}(c)$ and they can all do so in at most one step. Adding an additional constant d does suffice: $\mathbf{g}(c) \leftarrow \mathbf{g}(a) \rightarrow \mathbf{g}(d)$ with different ground normal forms $\mathbf{g}(c)$ and $\mathbf{g}(d)$. The same behaviour is observed for COM by noting that a TRS is (ground) confluent if and only if it (ground) commutes with itself.

The results in this section are summarized in Table 3, which shows the number of additional constants needed to reduce a property to the corresponding property on ground terms. In parentheses are the numbers for monadic TRSs.

For termination (SN) and normalization (WN) there is no need to add fresh constants, since these properties hold if and only if they hold for all ground terms. For other properties that can be expressed in the first-order theory of rewriting, one or two fresh constants may be insufficient. Consider for instance 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 arbitrary terms (with respect to any left-linear right-ground TRS $(\mathcal{F}, \mathcal{R})$). For the TRS consisting of the rule $f(x) \rightarrow a$ and two additional constants c and d , φ does not hold for ground terms because every ground term is convertible to a, c or d . It is tempting to believe that adding a fresh unary symbol g in addition to a fresh constant c , in order to create infinitely many ground normal forms which can replace variables that appear in open

terms, is sufficient for any property P . The formula $\forall s \forall t (s \rightarrow t \implies s \rightarrow_{\varepsilon} t)$ and the TRS consisting of the rule $a \rightarrow b$ show that this is incorrect.

7 Automation and Certification

7.1 Decision Mode

FORT-h is a new decision tool for the first order theory of rewriting. It is a reimplementation of the decision mode of the previous FORT tool [48], referred to as FORT-j in the remainder of the paper. The decision procedure implemented in FORT-j is based on the original procedure described in [10, 11], in which the basic relations are one-step and parallel rewriting. Anchored GTTs, which form the backbone of the formalized decision procedure described in this paper and implemented in FORT-h, were developed later. The new tool is implemented in Haskell whereas FORT-j is written in Java. FORT-h supports all features of FORT-j while extending the domain of supported TRSs from left-linear right-ground TRSs to *linear variable-separated* ones. While FORT-j could technically take such TRSs as input, it is unsound when checking non-ground properties on them.

Example 28 To check confluence of the linear variable-separated TRS

$$g(g(x)) \rightarrow g(y) \qquad a \rightarrow g(a)$$

FORT-h can be called with the formula CR. It correctly states that NO the system is not confluent. However, FORT-j incorrectly identifies this as confluent due to the lack of support for variables appearing in right-hand sides of rules.

FORT-h took part in the 2020, 2021 and 2022 editions of the Confluence Competition (CoCo),⁴ competing in five categories: COM, GCR, NFP, UNC and UNR. In 2021 and 2022 it also competed together with FORTify in the categories COM, TRS, GCR, UNC, UNR and NFP (only in 2022) producing certified answers. Even though it does not support many problems tested in the competition, due to the restriction to linear variable-separated TRSs, it was able to win the category for most YES results in UNR in all three years. The tool expects as input a formula and one or more TRSs, as seen in Fig. 7. It then outputs the answer YES or NO depending on whether the formula is satisfied or not by the given TRSs. The command-line interface of FORT-h is described in Appendix B.

The implemented procedure closely follows the procedure described in Sect. 5.5. When called it first parses the formula (format described below) and converts it into an internal representation using de Bruijn indices as described in Sect. 7.2. Additionally, universal quantifiers and implications are eliminated, and negations are pushed as far as possible to the atomic subformulas. The tool then traverses the formula in a bottom-up fashion, constructing the corresponding anchored GTTs and RR_n automata. During this traversal we also keep track of the steps taken, to construct the certificate if necessary. To improve performance the automata are cached and reused for equal subformulas. The tree automaton representing the whole formula is then checked for emptiness. If the accepted language is empty, FORT-h reports NO, otherwise it outputs YES.

To avoid having to write formulas using de Bruijn indices when using FORT-h, we use a more convenient syntax for interacting with the tool. The input format (later called FORT syntax) is described in Appendix A.

⁴ <http://project-coco.uibk.ac.at/>

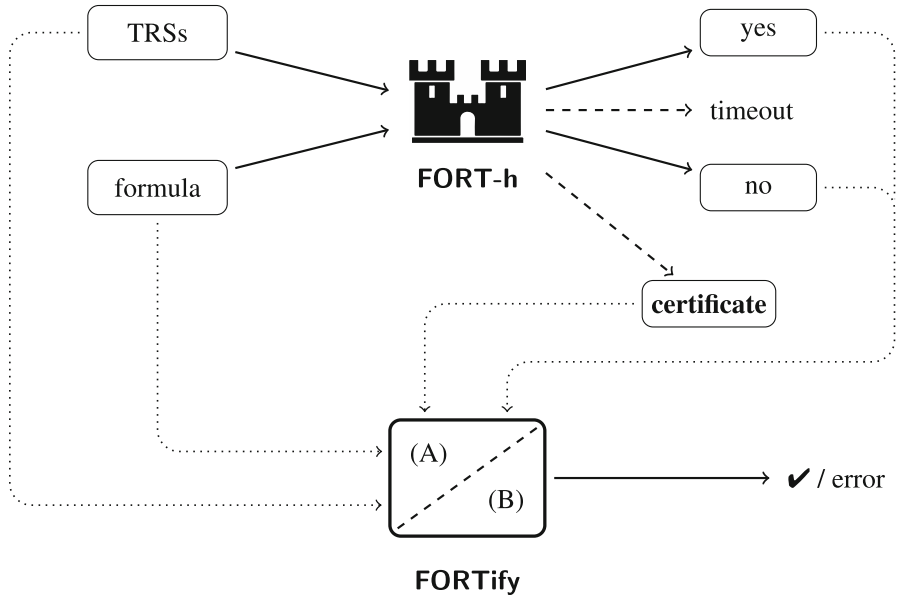


Fig. 7 FORT-h and FORTify

7.1.1 Witness Generation

The usual output of FORT-h consists of a YES or NO answer, and possibly a certificate containing size information of the automata. To help the user in understanding why a property holds or does not hold we support witness generation. This is possible in two cases. Firstly for *satisfiable existentially quantified* formulas, where FORT-h can produce an n -tuple of ground terms as evidence of existence. Secondly for *unsatisfiable universally quantified* formulas, where the tuple presents a counterexample. For instance, if a given or synthesized TRS is not ground-confluent $\neg \forall s \forall t \forall u (s \rightarrow^* t \wedge s \rightarrow^* u \implies \exists v (t \rightarrow^* v \wedge u \rightarrow^* v))$, it is interesting to provide witnessing terms for the variables s, t , and u . Given the TRS consisting of the rules

$$a \rightarrow f(a, b) \qquad f(a, b) \rightarrow f(b, a)$$

FORT-h produces the following terms as witnesses: $s = f(a, b)$, $t = f(b, a)$, and $u = f(f(a, b), b)$. To find these ground terms FORT-h first eliminates universal quantifiers using $\forall = \neg \exists \neg$, pushes negations inwards and removes double negations in the formula resulting in $\exists s \exists t \exists u (s \rightarrow^* t \wedge s \rightarrow^* u \wedge \neg \exists v (t \rightarrow^* v \wedge u \rightarrow^* v))$. In the next step FORT-h strips outermost negations, none in this case, followed by outermost existential quantifiers resulting in the so-called *formula body*: $(s \rightarrow^* t \wedge s \rightarrow^* u \wedge \neg \exists v (t \rightarrow^* v \wedge u \rightarrow^* v))$. Since the original formula is satisfiable, the RR_n automaton corresponding to the formula body must accept at least one n -tuple of ground terms.

The algorithm depicted in Fig. 8 generates (encoded) witnesses that are accepted by the given RR_n automaton. To find minimal witnesses we use a version of Dijkstra’s shortest path algorithm. We keep track of visited states in Q_v , a mapping \mathcal{W} from states to terms where $\mathcal{W}(q)$ is a minimal witness which reaches the state q , and a priority queue \mathcal{P} . The search is started at the states reachable in a single step from some constant. We also map from these

```

input: •  $RR_n$  automaton  $\mathcal{A} = (\mathcal{F}^{(n)}, Q, Q_f, \Delta)$ 
output: • term accepted by  $\mathcal{A}$ 
 $Q_v := \emptyset; \quad \mathcal{W} := \emptyset; \quad \mathcal{P} := \{(q, c_1 \cdots c_m, \text{size}(c_1 \cdots c_m)) \mid c_1 \cdots c_m \rightarrow q \in \Delta\};$ 
while  $\mathcal{P} \neq \emptyset$  do
  select  $(q, w, n) \in \mathcal{P}$  with minimal  $n$  and remove it from  $\mathcal{P}$ ;
  if  $q \in Q_f$  then return  $w$ ;
  else if  $q \notin Q_v$  then
     $Q_v := Q_v \cup \{q\}; \quad \mathcal{W} := \mathcal{W} \cup \{q \mapsto w\}; \quad \mathcal{P} := \mathcal{P} \cup \{(p, w, n) \mid q \rightarrow p \in \Delta\};$ 
    forall  $f_1 \cdots f_k(q_1, \dots, q_m) \rightarrow p \in \Delta$  where  $q \in \{q_1, \dots, q_m\} \subseteq Q_v$  do
       $w' := f_1 \cdots f_k(\mathcal{W}(q_1), \dots, \mathcal{W}(q_m));$ 
       $\mathcal{P} := \mathcal{P} \cup \{(p, w', \text{size}(w'))\};$ 
fail

```

Fig. 8 Witness generation

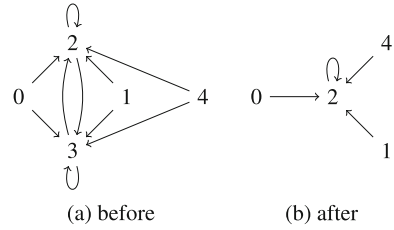
states to the respective constants as witnesses in \mathcal{W} . In each iteration we select the state q with the smallest witness w from \mathcal{P} . The size of a witness is determined by the function $\text{size}(\langle w_1, \dots, w_n \rangle) = \text{size}(w_1) + \dots + \text{size}(w_n)$, where $\text{size}(w_i)$ is the total number of function symbols in \mathcal{F} occurring in w_i , so \perp is not counted. If q is a final state we have found an accepted term and return the witness w . Otherwise we check that we have not visited q previously, set $\mathcal{W}(q) = w$, and enumerate all transition rules containing q on the left-hand side where all states on the left-hand side have been visited, and hence have a witness. If the transition rule is an epsilon transition $q \rightarrow p$, then the state p has the same witness as q so we add $(p, w, \text{size}(w))$ to \mathcal{P} . For a transition rule $f_1 \cdots f_k(q_1, \dots, q_m) \rightarrow p$ we construct a witness $w' = f_1 \cdots f_k(\mathcal{W}(q_1), \dots, \mathcal{W}(q_m))$ and add $(p, w, \text{size}(w))$ to the queue. The search continues until a final state is reached or all reachable states have been visited. In the latter case the algorithm fails, since the automaton does not accept any terms.

7.1.2 Collapsing ϵ -transitions

Keeping the size of automata small is crucial for the performance of FORT-h. One way to reduce the number of states and transitions is based on the observation that when two states q and p are strongly connected by ϵ -transitions, which means $q \xrightarrow{\epsilon}^* p$ and $p \xrightarrow{\epsilon}^* q$, then they are equivalent. In other words, for all ground terms s and t we have $s \xrightarrow{*} q$ if and only if $t \xrightarrow{*} p$, and for all ground contexts C and states r we have $C[q] \xrightarrow{*} r$ if and only if $C[p] \xrightarrow{*} r$. We can therefore replace all occurrences of a state in the transition rules by an equivalent one without changing the accepted language. This reduces the number of states, and may remove duplicate transition rules.

In FORT-h we can further take advantage of the fact that some of the most common constructions already produce sets of ϵ -transitions which are transitively closed. Instead of constructing the strongly connected components, checking if two states q and p are strongly connected then boils down to checking if $q \xrightarrow{\epsilon} p$ and $p \xrightarrow{\epsilon} q$. For example, this is case after computing the transitive closure of anchored GTT relations as in the Theorems 6 and 8. We therefore immediately collapse and eliminate the ϵ -transitions in the underlying tree automata after these constructions.

Fig. 9 Collapsing ε -transitions in $\Delta_+(\mathcal{A}, \mathcal{B})$



Example 29 The anchored GTT $G = (\mathcal{A}, \mathcal{B})$ with

$$\begin{array}{lll} \Delta_{\mathcal{A}}: & a \rightarrow 0 & b \rightarrow 1 \\ & 0 \rightarrow 3 & 1 \rightarrow 2 & 1 \rightarrow 4 \\ \Delta_{\mathcal{B}}: & a \rightarrow 2 & b \rightarrow 3 & c \rightarrow 4 \end{array}$$

accepts the rewrite relation of the ARS $\{a \rightarrow b, b \rightarrow a, b \rightarrow c\}$. When constructing $G_+ = (\mathcal{A} \cup \Delta_+(\mathcal{B}, \mathcal{A}), \mathcal{B} \cup \Delta_+(\mathcal{A}, \mathcal{B}))$, we need to compute the ε -transitions in $\Delta_+(\mathcal{A}, \mathcal{B})$. The result is shown in Fig. 9(a). We can see that the graph contains one non-trivial strongly connected component, consisting of the states $\{2, 3\}$. Instead of adding all 10 ε -transitions we can therefore simplify G and Δ_+ beforehand by replacing all occurrences of state 3 by state 2. This reduces the number of transitions in $\Delta_+(\mathcal{A}, \mathcal{B})$ to 4, as shown in Fig. 9(b), which, when added to G , results in the GTT $G_+ = (\mathcal{A}', \mathcal{B}')$ with

$$\begin{array}{lll} \Delta_{\mathcal{A}'}: & a \rightarrow 0 & b \rightarrow 1 \\ & 0 \rightarrow 2 & 1 \rightarrow 2 & 1 \rightarrow 4 & 2 \rightarrow 0 & 2 \rightarrow 4 & 2 \rightarrow 1 \\ \Delta_{\mathcal{B}'}: & a \rightarrow 2 & b \rightarrow 2 & c \rightarrow 4 \\ & 0 \rightarrow 2 & 4 \rightarrow 2 & 1 \rightarrow 2 \end{array}$$

Note that we also dropped the redundant transition $2 \rightarrow 2$ from $\Delta_+(\mathcal{A}, \mathcal{B})$.

7.2 Certification

Whereas witness generation can only provide some evidence to assist the user in understanding why certain formulas hold or not, in certification we are interested in machine-readable proofs that are verified by an independent and trustworthy certifier. The first step in the certification process is to translate formulas in the first-order theory of rewriting into a format suitable for further processing. We adopt de Bruijn indices [13] to avoid alpha renaming.

Example 30 Consider the formula

$$\forall s \forall t \forall u (s \rightarrow_0^* t \wedge s \rightarrow_1^* u \implies \exists v (t \rightarrow_1^* v \wedge u \rightarrow_0^* v))$$

It expresses the *commutation* of two TRSs, indicated by the indices 0 and 1. Using de Bruijn indices for the term variables s, t, u, v produces

$$\forall \forall \forall (2 \rightarrow_0^* 1 \wedge 2 \rightarrow_1^* 0) \implies \exists (2 \rightarrow_1^* 0 \wedge 1 \rightarrow_0^* 0)$$

We refer to Example 32 for further explanation.

The formal syntax of formulas in certificates is given below. Here $\langle rr_2 \rangle$ denotes the supported binary regular relations, which are formally defined after Example 31. Likewise,

$\langle rr_1 \rangle$ stands for regular sets (which are identified with unary regular relations).

```

(formula) ::= (rr1 <rr1> <term>) | (rr2 <rr2> <term> <term>)
           | (and <formula> *) | (or <formula> *) | (not <formula>)
           | (forall <formula>) | (exists <formula>) | (true) | (false)
           | (restrict <formula> (<trs> +))
(term) ::= <nat> <trs> ::= <nat> | <nat> - <nat> ::= 0 | 1 | 2 | ...
    
```

De Bruijn indices are used for $\langle term \rangle$ variables and $\langle nat \rangle$ – denotes a TRS with index $\langle nat \rangle$ in which the left- and right-hand sides of the rules have been swapped. The class of linear variable-separated TRSs is closed under this operation. We use it to represent the conversion relation \leftrightarrow^* of a TRS \mathcal{R} as the reachability relation \rightarrow^* induced by the TRS $\mathcal{R} \cup \mathcal{R}^-$.

Example 31 The commutation property in Example 30 is rendered as follows:

```

(forall (forall (forall (or (not (and (rr2 (step* (0)) 2 1)
    (rr2 (step* (1)) 2 0))) (exists (and (rr2 (step* (1)) 2 0)
    (rr2 (step* (0)) 1 0))))))
    
```

Here $(step^* (0))$ denotes the RR_2 relation \rightarrow^* induced by the first TRS (which is indexed by 0) and $(rr2 (step^* (1)) 2 0)$ represents the subformula $[1] \text{ t } \rightarrow^* \text{ v}$ of the FORT formula in Example 30.

We continue with the certificate syntax of RR_1 and RR_2 relations:

```

<rr1> ::= (terms) | (nf (<trs> +)) | (inf <rr2>) | (proj (1|2) <rr2>)
       | (union <rr1> <rr1>) | (inter <rr1> <rr1>) | (diff <rr1> <rr1>)
<rr2> ::= (gtt <gtt> <pos> <num>) | (product <rr1> <rr1>) | (id <rr1>)
       | (union <rr2> <rr2>) | (inter <rr2> <rr2>) | (diff <rr2> <rr2>)
       | (comp <rr2> <rr2>) | (inverse <rr2>)
<pos> ::= >= | e | > <num> ::= >= | 1 | >
<gtt> ::= (root-step (<trs> +)) | (gsteps (<trs> +)) | (inverse <gtt>)
       | (union <gtt> <gtt>) | (acomp <gtt> <gtt>) | (gcomp <gtt> <gtt>)
       | (inter <gtt> <gtt>) | (acomplement <gtt>) | (atc <gtt>) | (gtc <gtt>)
    
```

Here $(terms)$ refers to $\mathcal{T}(\mathcal{F})$, $(nf (<trs> +))$ to the normal forms (NF) induced by the union of the underlying TRSs, and $(inf <rr_2>)$ to the infinity predicate (INF_R) which is satisfied by all terms having infinitely many successors with respect to the relation R . Furthermore, $(proj (1|2) <rr_2>)$ denotes projection (π) to the first (second) argument, $(gtt <gtt> <pos> <num>)$ the transformation of a GTT relation into an RR_2 relation with corresponding context closure (Theorems 10 and 11), $(id <rr_1>)$ the identity relation on the underlying set, and $(gtc <gtt>) ((atc <gtt>))$ the (anchored) transitive closure of the underlying (anchored) GTT relation. The $(gsteps (<trs> +))$ construct serves as an abbreviation for $(gtc ((root-step (<trs> +)))$.

The constructs defined above closely correspond to the formalized closure operations for the predicates in the first-order theory of rewriting, summarized in the grammar in Fig. 1.

For convenience of tool authors, we add a few other constructs to $\langle rr_2 \rangle$. The certifier expands these to a sequence of basic constructs given above.

$$\begin{aligned} \langle rr_2 \rangle ::= & \dots \mid (\text{step} (\langle trs \rangle +)) \mid (\text{step} = (\langle trs \rangle +)) \mid (\text{step} + (\langle trs \rangle +)) \\ & \mid (\text{step}^* (\langle trs \rangle +)) \mid (\text{step}! (\langle trs \rangle +)) \mid (\text{equality}) \\ & \mid (\text{parallel-step} (\langle trs \rangle +)) \mid (\text{root-step} (\langle trs \rangle +)) \\ & \mid (\text{root-step} = (\langle trs \rangle +)) \mid (\text{root-step} + (\langle trs \rangle +)) \\ & \mid (\text{root-step}^* (\langle trs \rangle +)) \mid (\text{non-root-step} (\langle trs \rangle +)) \\ & \mid (\text{non-root-step} = (\langle trs \rangle +)) \mid (\text{non-root-step} + (\langle trs \rangle +)) \\ & \mid (\text{non-root-step}^* (\langle trs \rangle +)) \mid (\text{meet} (\langle trs \rangle +)) \\ & \mid (\text{join} (\langle trs \rangle +)) \mid (\text{reflcl} (\langle rr_2 \rangle)) \end{aligned}$$

A certificate for a first-order formula φ explains how the corresponding RR_n automaton is constructed. We adopt a line-oriented natural deduction style. The automata are implicit. This is a deliberate design decision to keep certificates small. More importantly, it avoids having to check equivalence of finite tree automata, which is EXPTIME-complete [8, Sect. 1.7].

$$\begin{aligned} \langle certificate \rangle ::= & (\langle item \rangle \langle inference \rangle \langle formula \rangle \langle info \rangle *) \langle certificate \rangle \\ & \mid (\text{empty} (\langle item \rangle)) \mid (\text{nonempty} (\langle item \rangle)) \\ \langle item \rangle ::= & \langle nat \rangle \quad \langle info \rangle ::= (\text{size} \langle nat \rangle \langle nat \rangle \langle nat \rangle) \\ \langle inference \rangle ::= & (\text{rr1} (\langle rr_1 \rangle \langle term \rangle)) \mid (\text{rr2} (\langle rr_2 \rangle \langle term \rangle \langle term \rangle)) \mid (\text{and} (\langle item \rangle *) \\ & \mid (\text{or} (\langle item \rangle *) \mid (\text{not} (\langle item \rangle)) \mid (\text{exists} (\langle item \rangle)) \mid (\text{nnf} (\langle item \rangle)) \end{aligned}$$

Currently the $\langle info \rangle$ field only serves as an interface between the tool (which provides the certificate) and the certifier to compare the sizes of the constructed automata. In the future we plan to extend this field with concrete automata. This allows to test language equivalence of a tree automaton computed by a tool that supports our certificate language and the one reconstructed by FORTify, thereby providing tool authors with a mechanism to trace buggy constructions in case a certificate is rejected.

We revisit Example 3 to illustrate the construction of certificates.

Example 32 The formula $\varphi = \forall s \exists t (s \rightarrow^* t \wedge \text{NF}(t))$ expressing normalization is rendered as $\varphi' = \forall \exists (1 \rightarrow_0^* 0 \wedge 0 \in \text{NF}[0])$ in de Bruijn notation. Here 1 refers to the variable s , the second and third occurrences of 0 refer to t , and the last occurrence of 0 refer to the first (and only) TRS, which has index 0. We construct the certificate bottom-up, to mimic the decision procedure. The first line is for $\text{NF}[0]$:

$$(0 (\text{rr1} (\text{nf} (0)) 0) (\text{rr1} (\text{nf} (0)) 0))$$

The components can be read as follows:

- $\langle item \rangle = 0$ denotes the first step in our proof,
- $\langle inference \rangle = \text{rr1} (\text{nf} (0)) 0$ constructs the automaton that accepts the normal forms and keeps track of the variable 0,
- $\langle formula \rangle = \text{rr1} (\text{nf} (0)) 0$ denotes the subformula $0 \in \text{NF}[0]$; it is satisfiable if and only if the automaton constructed using the description in $\langle inference \rangle$ is not empty.

The apparent redundancy will disappear when we continue. We proceed by expressing the relation \rightarrow_0^* and subsequently make sure that the second component of \rightarrow_0^* is in normal form:

```
(1 (rr2 (step* (0)) 1 0) (rr2 (step* (0)) 1 0))
(2 (and (1 0)) (and ((rr2 (step* (0)) 1 0) (rr1 (nf (0)) 0))))
```

Line 1 is similar to line 0. The inference step `(and 1 0)` in line 2 constructs an RR_2 automaton that accepts the intersection of the relations modeled in lines 1 and 0. This automaton corresponds to \mathcal{A}_5 in Example 3. The cylindrification step from \mathcal{A}_1 to \mathcal{A}_4 in Example 3 is left implicit. We continue with the projection of variable 0 and afterwards complement the resulting automaton. This is done by an `exists` followed by a `not` inference step:

```
(3 (exists 2) (exists (and ((rr2 (step* (0)) 1 0)
  (rr1 (nf (0)) 0))))))
(4 (not 3) (not (exists (and ((rr2 (step* (0)) 1 0)
  (rr1 (nf (0)) 0))))))
```

The inference steps until this point describe the construction of \mathcal{A}_7 in Example 3. We complete the certificate by introducing the remaining operators:

```
(5 (exists 4) (exists (not (exists (and ((rr2 (step* (0)) 1 0)
  (rr1 (nf (0)) 0)))))))
(6 (not 5) (not (exists (not (exists (and ((rr2 (step* (0)) 1 0)
  (rr1 (nf (0)) 0)))))))
(7 (nnf 6) (forall (exists (and ((rr2 (step* (0)) 1 0)
  (rr1 (nf (0)) 0))))))
(nonempty 7)
```

The `nnf` inference step does not modify the tree automaton computed in step 6 (which corresponds to \mathcal{A}_9 in Example 3) but checks the equivalence of the formula in line 6 with the one of line 7, which corresponds to the input formula φ' . The equivalence check incorporates \forall elimination, negation normal form, and associativity, commutativity and idempotency of \wedge and \vee . In the future we might add support for additional equivalences in first-order logic. The final step `(nonempty 7)` checks that $L(\mathcal{A}_9) \neq \emptyset$. So this certificate claims that the input TRS is normalizing. For TRSs that do not satisfy φ , the final line in the certificate would be `(empty 7)`.

In the previous example we intentionally skipped over some details to convey the underlying intuition. First of all, the `(rr2)` construct `(step* (0))` is derived and internally unfolded via (anchored) GTTs into

```
(gtt (gtc (root-step 0)) >= >)
```

Starting from an anchored GTT that accepts the root step relation induced by the first (and only) TRS in the list, an application of the GTT transitive closure operation followed by a multi-hole context closure operation with at least one hole that may appear in any position, an RR_2 automaton that accepts the relation \rightarrow_0^* is constructed. We also mentioned that cylindrification is implicit. The same holds for the projection operation that is used in the `exists` inference steps. A projection takes place in the first component if the variable 0 is present in the list of variables, otherwise the inference step preserves the automaton. This approach is sound as variables indicate the relevant components of the RR_n automaton. Thanks to the de Bruijn representation, the innermost quantifier refers to variable 0, the first component in the given RR_n automaton. However we must keep track of all variables occurring in the surrounding formula and update that list accordingly.

7.3 FORTify

The example in the preceding subsection makes clear that certificate can be viewed as a recipe for the certifier to perform certain operations on automata and formulas to confirm the final (non-)emptiness claim. In particular, checking a certificate is expensive because the decision procedure for the first-order theory is replayed using code-generated operations from a verified version of the decision procedure. In this subsection we describe the steps we performed to turn the Isabelle formalization of the decision procedure into our certifier FORTify.

The formalization is split into two parts. The second part is about the certification process, but we start our description with the first part [35] which serves as a general tree automata library. This part includes bottom-up tree automata with ϵ -transitions, (anchored) ground tree transducers, encoding of regular relations, and their respective closure properties. Additionally it contains a framework to simplify code generation of inductively defined sets as in Fig. 3. Such inductive sets, if they are finite, can be computed by a saturation procedure. We provide an abstraction for that, which essentially does Horn inference without negative atoms. The point of the abstraction is that it separates a common iterative or recursive part of saturation procedures (which gives rise to non-trivial correctness proofs) from the enumeration of inferences without premises (\mathcal{H}_0 , see below), and inferences induced by a single new conclusion (\mathcal{H}_1 , also below), which usually are set comprehensions that can be computed in a very straightforward way.

Definition 19 A positive Horn inference system is given by a set of atoms A (with elements α, β, \dots) and set \mathcal{H} of inference rules of the shape $\alpha_1 \wedge \dots \wedge \alpha_n \rightarrow \beta$. We write $\top \rightarrow \beta$ if the list of premises is empty. Each positive Horn inference system defines a predicate $\overline{\mathcal{H}}$ on atoms inductively by the rule

$$\frac{\alpha_1 \wedge \dots \wedge \alpha_n \rightarrow \beta \in \mathcal{H} \quad \overline{\mathcal{H}}(\alpha_i) \text{ for } 1 \leq i \leq n}{\overline{\mathcal{H}}(\beta)}$$

Example 33 Consider the inference rules from Fig. 3. To obtain a positive Horn inference system for given automata \mathcal{A} and \mathcal{B} , let $A = Q \times Q$ where Q is the set of states occurring in \mathcal{A} or \mathcal{B} . The set \mathcal{H} consists of the following inference rules:

- $(p, r) \rightarrow (q, r)$ if $p \rightarrow_{\mathcal{A}} q$ and $r \in Q$,
- $(p, q) \rightarrow (p, r)$ if $q \rightarrow_{\mathcal{B}} r$ and $p \in Q$, and
- $(p_1, q_1) \wedge \dots \wedge (p_n, q_n) \rightarrow (p, q)$ if $f(p_1, \dots, p_n) \rightarrow_{\mathcal{A}} p$ and $f(q_1, \dots, q_n) \rightarrow_{\mathcal{B}} q$.

These Horn clauses correspond directly to Fig. 3 with $p \rightsquigarrow q$ replaced by (p, q) . It is easy to see that the resulting $\overline{\mathcal{H}}$ satisfies $(p, q) \in \overline{\mathcal{H}}$ if and only if $p \rightsquigarrow q$.

We have formalized an abstract marking algorithm for positive Horn inference systems. In order to use this algorithm, the user has to provide implementations for two building blocks, \mathcal{H}_0 and \mathcal{H}_1 , which are given by

$$\mathcal{H}_0 = \{\beta \mid \top \rightarrow \beta \in \mathcal{H}\}$$

$$\mathcal{H}_1(\alpha, B) = \{\beta \mid \alpha_1 \wedge \dots \wedge \alpha_n \rightarrow \beta \in \mathcal{H} \text{ and } \alpha \in \{\alpha_1, \dots, \alpha_n\} \subseteq B \cup \{\alpha\}\}$$

In essence, \mathcal{H}_0 computes inferences without premises, whereas $\mathcal{H}_1(\alpha, B)$ provides all possible conclusions involving a particular premise α together with other premises fulfilled by B . These two ingredients are sufficient to implement a simple marking algorithm:

```

saturate_rec( $\alpha, I$ ):
  if  $\alpha \in I$  then return  $I$ 
  else
     $J := \{\alpha\} \cup I$ ;
    for all  $\beta \in \mathcal{H}_1(\alpha, I)$  do
       $J := \text{saturate\_rec}(\beta, J)$ ;
    return  $J$ 

saturate:
   $I := \emptyset$ ;
  for all  $\alpha \in \mathcal{H}_0$  do
     $I := \text{saturate\_rec}(\alpha, I)$ 
  return  $I$ 
    
```

Most of the work is performed by `saturate_rec`, whose purpose is to add a newly inferred atom α to an accumulator I of previously inferred atoms, taking into account all further inferences that can be made using α and elements of I . It relies on \mathcal{H}_1 for computing the set of atoms that can be inferred using β at least once and elements of I for other premises. The main method `saturate` iterates over the elements of \mathcal{H}_0 and adds them to the accumulator I using the `saturate_rec` helper, starting with $I = \emptyset$. We formalized soundness of `saturate`, and of refinements to lists and finite sets.

Example 34 Continuing from Example 33, we note that the computation of \mathcal{H}_0 and \mathcal{H}_1 can often be done efficiently without ever computing the full set \mathcal{H} . For the inference rules from Fig. 3, we obtain the following descriptions:

$$\mathcal{H}_0 = \{(p, q) \mid f \rightarrow_{\mathcal{A}} p \text{ and } f \rightarrow_{\mathcal{B}} q\}$$

$$\mathcal{H}_1((p, q), B) = \{(r, q) \mid p \rightarrow_{\mathcal{A}} r\} \cup \{(p, r) \mid q \rightarrow_{\mathcal{B}} r\} \cup \mathcal{H}'_1$$

where \mathcal{H}'_1 consists of all pairs (p', q') such that

$$f(p_1, \dots, p_n) \rightarrow_{\mathcal{A}} p' \qquad f(q_1, \dots, q_n) \rightarrow_{\mathcal{B}} q'$$

with $(p_i, q_i) \in B \cup \{(p, q)\}$ for all $1 \leq i \leq n$, and $(p, q) = (p_i, q_i)$ for some $1 \leq i \leq n$. This last component is slightly complicated (but not much more complicated than the definition of \mathcal{H} itself). On the other hand, the first two components of \mathcal{H}_1 make no reference to Q , which is a welcome simplification.

Isabelle/HOL has a *predicate compiler* [5] that produces executable code for certain inductive sets, but it is quite restricted; basically, it works by searching all possible derivation trees to arrive at a conclusion. This easily leads to non-termination when there are infinitely many such trees, which often happens. For example, using the rules in Fig. 3, if we want to check whether $1 \rightsquigarrow 2$ and there is an ε -transition $1 \rightarrow_{\mathcal{A}} 1$, then the first inference rule is a possible candidate for the last inference step, leading us to check $1 \rightsquigarrow 2$ recursively, ad infinitum.

In our formalization, GTT compositions and GTT transitive closure are implemented on top of positive Horn inference. The other building blocks are derived directly from the definitions, using automatic and some manual refinement to obtain concrete implementations.

This concludes the first part. In the remainder of this section details of the second part are discussed [33]. We use the FOL-Fitting library [4], which is part of the Archive of Formal Proofs, to connect the first-order theory of rewriting and first-order logic. The translation is more or less straightforward. We interpret RR_1 constructions as predicates and RR_2 constructions as relations in first-order logic and prove both interpretations to be semantically equivalent:

lemma *eval_formula* $\mathcal{F} \text{ Rs } \alpha f =$
eval α *undefined* (for_eval_rel $\mathcal{F} \text{ Rs}$) (form_of_formula f)

With this equivalence we are able to define the semantics of formulas:

definition *formula_satisfiable* **where**

$$\begin{aligned} \text{formula_satisfiable } \mathcal{F} \text{ Rs } f &\longleftrightarrow (\exists \alpha. \text{range } \alpha \subseteq T_G \mathcal{F} \wedge \\ \text{eval_formula } \mathcal{F} \text{ Rs } \alpha f) \end{aligned}$$

definition *formula_unsatisfiable* **where**

$$\text{formula_unsatisfiable } \mathcal{F} \text{ Rs } fm \longleftrightarrow (\text{formula_satisfiable } \mathcal{F} \text{ Rs } fm = \text{False})$$

definition *correct_certificate* **where**

$$\begin{aligned} \text{correct_certificate } \mathcal{F} \text{ Rs } \text{claim infs } n &\equiv \\ (\text{claim} = \text{Empty} &\longleftrightarrow (\text{formula_unsatisfiable } (fset \mathcal{F}) (\text{map } fset \text{ Rs}) \\ &(\text{fst } (snd (snd (infs ! n)))))) \wedge \\ \text{claim} = \text{Nonempty} &\longleftrightarrow \text{formula_satisfiable } (fset \mathcal{F}) (\text{map } fset \text{ Rs}) \\ &(\text{fst } (snd (snd (infs ! n)))) \end{aligned}$$

Last but not least we define the important function `check_certificate` which takes as input a signature, a list of TRSs, a Boolean, a formula, and a certificate. This function first verifies that the given formula and the claim corresponds to the ones referenced in the certificate and afterwards checks the integrity of the certificate. The following lemmata, which are formally proved in Isabelle, state the correctness of the `check_certificate` function:

$$\begin{aligned} \text{lemma } \text{check_certificate } \mathcal{F} \text{ Rs } A \text{ fm } (\text{Certificate infs claim } n) = \text{Some } B \\ \implies fm = \text{fst } (snd (snd (infs ! n))) \wedge A = (\text{claim} = \text{Nonempty}) \end{aligned}$$

$$\begin{aligned} \text{lemma } \text{check_certificate } \mathcal{F} \text{ Rs } A \text{ fm } (\text{Certificate infs claim } n) = \text{Some } B \\ \implies (B = \text{True} \longrightarrow \text{correct_certificate } \mathcal{F} \text{ Rs } \text{claim infs } n) \wedge \\ (B = \text{False} \longrightarrow \text{correct_certificate } \mathcal{F} \text{ Rs } (\text{case claim of} \\ \text{Empty} \Rightarrow \text{Nonempty} \mid \text{Nonempty} \Rightarrow \text{Empty}) \text{ infs } n) \end{aligned}$$

The first lemma ensures that our check function verifies that the provided parameters *fm* (formula) and *A* (answer satisfiable/unsatisfiable) match the formula and the claim stated in the certificate. The second lemma is the key result. It states that the check function returns `Some True` if and only if the certificate is correct. The only-if case is hidden in the last two lines. More precisely, if the claim of the certificate is wrong then negating the claim (the first-order theory of rewriting is complete) leads to a correct certificate. Therefore, if our check function returns `Some None` then the certificate is correct after negating the claim.

Our check function returns `None` if the global assumptions (the input TRS is not linear variable-separated, the signature is not empty, etc.) are not fulfilled. We plan to extend the `check_certificate` function in the near future such that it reports these kinds of errors.

A central part of the formalization is to obtain a trustworthy decision procedure to verify certificates. Hence we use the code generation facility of Isabelle/HOL to produce an executable version of our `check_certificate` function. Isabelle's code generation facility is able to derive executable code for our constructions with the exception of inductively defined sets. We use the abstract Horn inference system framework of Definition 19 to obtain executable code for the following constructions defined as inductive sets:

- reachable and productive states of a tree automaton,

Table 4 Formalization statistics

Topics	Lines	Facts	Defs
Utility files	1892	187	19
Terms, context, and rewriting	3969	454	97
Horn inference system	462	39	17
Tree automata	2891	319	66
Regular relations	4016	285	65
Primitives and context closure	4043	318	43
FORT decision procedure	2023	107	60
Signature extension	2874	182	15
Implementation files	3058	190	81
Total	25, 228	2081	463

- states of tree automata obtained by the subset construction,
- ε -transitions for the composition and transitive closure constructions of (anchored) GTTs,
- an inductive set needed for the tree automaton for the infinity predicate.

At this point we can use Isabelle's code generation to obtain an executable check function. The resulting code-generated certifier is called FORTify.

The overall design of FORTify is shown in the bottom half of Fig. 7. It can be viewed as two separate modules A and B. Module B is the verified Haskell code base that is generated by Isabelle's code generation facility, containing the `check_certificate` function and the data type declarations for formulas and certificates. To use this functionality, we wrote a parser which translates strings representing formulas (signatures, TRSs, certificates) to semantically equivalent formulas (signatures, TRSs, certificates) represented in the data types obtained from the generated code. This was done in Haskell and refers to module A in Fig. 7. Module A accepts formulas in FORT syntax. Hence it also applies the conversion to the de Bruijn representation. After the translation in module A, the `check_certificate` function in module B is executed and its output is reported.

Importantly, the code in module A is not verified in Isabelle. Correctness of FORTify must therefore assume correctness of module A as well as the correctness of the Glasgow Haskell Compiler, which we use to generate a standalone executable from the generated code.

Table 4 lists some statistics of the underlying formalization.

7.4 Synthesis Mode

FORT can be used to synthesize TRSs that satisfy properties given by the user (which is different from finding witnessing terms in formulas as described in Sect. 7.1). This is useful for finding counterexamples and non-trivial TRSs for exam exercises as well as competitions. The synthesis procedure for a given signature \mathcal{F} boils down to generating candidate TRSs and then checking the given property as shown in Fig. 10. The latter is done using a call to the decision procedure `decide(\mathcal{F} , φ , C)`, which checks if the formula φ holds for the system C over the domain $\mathcal{T}(\mathcal{F})$. To limit and control the search space we introduce the parameters r , R , D and v :

- r and R specify the lower and upper bound on the number of rewrite rules,
- D specifies the upper bound on the height of the left- and right-hand sides of the rules,
- v specifies the number of different variables that may appear in the rewrite rules.

```

input:  • closed first-order formula  $\varphi$ , signature  $\mathcal{F}$ , parameters  $r R D v, n$ 
output: • TRS satisfying  $\varphi$ 
V := { $x_1, \dots, x_v$ }; T :=  $V \cup \{c \mid c \text{ is constant in } \mathcal{F}\}$ 
while D > 0 do
  S := { $\ell \rightarrow r \mid \ell, r \in T$  satisfying the rule constraints}      -- rewrite rules
  C' := { $C \subseteq S \mid C$  satisfies the  $r, R$  constraints}                -- candidate TRSs
  foreach C  $\in$  C' do
    if decide( $\mathcal{F}, \varphi, C$ ) succeeds then return C                    -- call to decision mode
  T' := { $f(t_1, \dots, t_n) \mid f \in \mathcal{F}$  has arity  $n > 0, t_1, \dots, t_n \in T$ } -- new terms
  T :=  $T \cup T'$ 
  D :=  $D - 1$ 
return "no TRS found"

```

Fig. 10 Simplified synthesis procedure (for a fixed signature)

By default the procedure searches for left-linear right-ground TRSs, but can also synthesize linear variable-separated systems. This affects the generation of candidate TRSs S in Fig. 10.

To extend the functionality and improve performance, the implementation in the synthesis tool (FORT-s) differs from the procedure in Fig. 10. Since the greatest cost when running the procedure comes from executing the decision procedure, care is taken to not generate and check equivalent system more than once. To this end, we keep track of fresh terms from previous iterations and only generate rules containing at least one new term, and the fresh terms in T' must contain at least one new term in an argument position. Similar improvements are used when generating the rewrite systems. The second major performance improvement is the possibility of checking systems in parallel.

It is of interest to synthesize TRSs that depend on one or more other TRSs. This can be done by passing additional TRSs to FORT-s in addition to a formula which references multiple systems. The additional systems are then also passed to the decision procedure. For example, if we want to transform our leading TRS \mathcal{R} (see Example 1) into an equivalent complete TRS (on ground terms), we pass both \mathcal{R} and the formula

$$(\text{GWCR}_0 \wedge \text{SN}_0) \wedge \forall s \forall t (s \leftrightarrow_0^* t \iff s \leftrightarrow_1^* t)$$

to FORT-s. Here the index 1 refers to \mathcal{R} and the index 0 to the system to be synthesized. This returns the TRS consisting of the rules

$$a \rightarrow b \qquad f(b) \rightarrow g(a, a) \qquad g(b, b) \rightarrow a$$

Using formulas referencing multiple TRSs FORT-s can also be used to synthesize multiple systems.

For convenience FORT-s supports multiple ways to specify the signature used during synthesis. The full user interface of FORT-s is given in Appendix C.

7.5 Undecidability of Synthesis

Since the first-order theory is decidable for linear variable-separated TRSs a natural question arises. Is synthesis also decidable for these systems? In other words, can we determine if there exists a linear variable-separated TRS satisfying a given property? Unfortunately this is not the case.

Theorem 17 *The following problem is undecidable:*

- instance:* a closed formula φ in the first-order theory of rewriting
- question:* does some linear variable-separated TRS \mathcal{R} satisfy φ

Proof We show the undecidability by a reduction from Post correspondence problem. Let P be a finite set of pairs of non-empty strings over the alphabet $\{0, 1\}$. We define a formula φ_P in the first-order theory of rewriting that is satisfiable if and only if P has a solution. To this end, consider the following predicates:

$$\begin{aligned}
 \text{node}(x) &:= x \rightarrow x \\
 \text{next}(x, y) &:= \text{node}(x) \wedge \text{node}(y) \wedge x \rightarrow y \wedge x \neq y \\
 \text{step} &:= \forall x (\text{node}(x) \wedge x \neq e \implies \exists y \text{next}(x, y)) \\
 \text{unique} &:= \forall x \forall y \forall z (\text{next}(x, y) \wedge \text{next}(x, z) \implies y = z) \\
 \text{linear} &:= \text{step} \wedge \text{unique} \\
 \text{value}(x, 0) &:= x \rightarrow a \wedge \neg(x \rightarrow b) \\
 \text{value}(x, 1) &:= x \rightarrow b \wedge \neg(x \rightarrow a) \\
 \text{finite} &:= \neg \exists x \text{INF}_{\neq}(x)
 \end{aligned}$$

Positions in a solution string are represented by *nodes*, which are linearly ordered. Nodes are characterized by self-loops. The special nodes s and e mark the starting and final positions in a solution of P . The predicate *finite* ensures that solution strings are finite. We have two additional elements, a and b that correspond to the symbols 0 and 1.

$$\text{border}(x, y) := \text{node}(x) \wedge \text{node}(y) \wedge \exists z (\neg \text{node}(z) \wedge x \rightarrow z \wedge z \rightarrow y)$$

The *border* predicate marks the two positions in a solution string corresponding to the decomposition into first and second components. The latter is checked by the *solution* predicate:

$$\begin{aligned}
 \text{match}(x_0, x_1 \cdots x_k, v_1 \cdots v_k) &:= \bigwedge_{i=1}^k (\text{next}(x_{i-1}, x_i) \wedge \text{value}(x_i, v_i)) \\
 \text{pair}(x, y, v, w) &:= \exists x_1 \dots \exists x_{|v|} \exists y_1 \dots \exists y_{|w|} (\text{border}(x_{|v|}, y_{|w|}) \wedge \\
 &\quad \text{match}(x, x_1 \cdots x_{|v|}, v) \wedge \text{match}(y, y_1 \cdots y_{|w|}, w)) \\
 \text{solution} &:= \forall x \forall y (\text{border}(x, y) \implies \\
 &\quad (x = y \wedge x = e) \vee \bigvee_{(v,w) \in P} \text{pair}(x, y, v, w))
 \end{aligned}$$

The formula φ_P is now defined as

$$\exists s \exists e \exists a \exists b (s \neq e \wedge \text{border}(s, s) \wedge \text{linear} \wedge \text{finite} \wedge \text{solution})$$

□

Note that the witnessing TRSs constructed in the above proof are actually abstract rewrite systems (ARSs) that consist of rewrite rules between constants. The construction is illustrated in Fig. 11, for the PCP instance $P = \{(1, 011), (10, 11), (001, 00)\}$ with solution $001|10|001|1 = 00|11|00|011$. The separation bars correspond to the elements b_1, b_2 and b_3 . Node n_9 witnesses e . Elements 0 and 1 witness a and b .

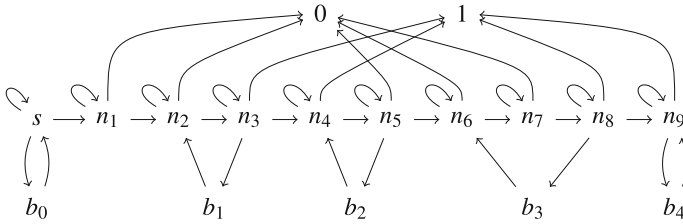


Fig. 11 The construction for PCP instance P

The synthesis problem is obviously decidable for ARSs over a fixed signature, but remains undecidable for TRSs over a fixed signature, since we can still generate an arbitrary number of ground terms using non-constant function symbols. Take for example the signature $\{E, s, 0\}$, where E and s are unary function symbols and 0 is a constant. We can then represent an arbitrary number n of objects (nodes, borders and values in the encoding) using ground terms of the shape $E(s^n(0))$. The rules of the ARS correspond to rules between such ground terms of the generated TRS. (The inclusion of the function symbol E removes any possibility of unwanted overlap between rules of the TRS.)

8 Experiments

In this section we describe the experiments we performed with FORT-h, FORT-s, and FORTify. We include version 1.0 of FORT-h, which was first published as part of an artifact⁵ in conjunction with [42]. The current version of FORT-h is 2.0. Full details of the experiments are available from the website⁶ accompanying this paper. Precompiled binaries of FORT-h 2.0, FORT-s, and FORTify are available from the same site. All experiments were run on a computer equipped with an Intel Core i7-5930K processor with 6 cores, and with 32 GB of memory. To remove any ambiguity in the calls made to the tools we use FORT-syntax (see Appendix A) to specify formulas in this section. This also aids in replicating the experiments.

8.1 FORT-h and FORTify

For the experiments reported in this section we used a timeout of 60 s for the decision tools and 600 s for FORTify.

8.1.1 Comparing Different Representations of Properties

The problems for these experiments are taken from the Confluence Problems database (COPS),⁷ and consists of 122 left-linear right-ground TRSs. The formulas were taken from the experiments reported in [46].

Experiment 1 The first three

$$\text{"forall } s, t, u \text{ (} s \text{ -> }^* t \text{ \& } s \text{ -> }^* u \text{ => } t \text{ join } u \text{)" } \quad (15)$$

⁵ <https://fortissimo.uibk.ac.at/tacas2021/>

⁶ <https://fortissimo.uibk.ac.at/jar>

⁷ <https://cops.uibk.ac.at/>

Table 5 FORT-h (with FORTify) and FORT-j run on GCR formulas

		YES	∅-time	✓	NO	∅-time	✓	∞	total (✓) time
(15)	FORT-h 2.0	37	0.89 s	37	84	0.69 s	81	1	151.12 s (0.8h)
	FORT-h 1.0	36	0.26 s	10	84	0.56 s	16	2	176.23 s (17.6h)
	FORT-j	37	0.31 s	–	82	0.52 s	–	3	234.08 s
(16)	FORT-h 2.0	38	1.50 s	37	84	0.06 s	81	0	62.13 s (0.9h)
	FORT-h 1.0	37	1.48 s	10	84	0.09 s	16	1	122.55 s (17.8h)
	FORT-j	37	0.32 s	–	82	0.50 s	–	3	233.20 s
(17)	FORT-h 2.0	37	0.91 s	37	83	0.04 s	81	2	156.64 s (1.0h)
	FORT-h 1.0	36	0.45 s	6	83	0.08 s	9	3	202.64 s (18.2h)
	FORT-j	37	0.32 s	–	82	0.55 s	–	3	236.69 s

$$\text{"forall } s, t, u (s \rightarrow^* t \ \& \ s \rightarrow u \Rightarrow t \text{ join } u) \text{"} \tag{16}$$

$$\text{"forall } t, u (t \leftarrow^* u \Rightarrow t \text{ join } u) \text{"} \tag{17}$$

denote different but equivalent formulations of ground-confluence (GCR). The results are shown in Table 5, where the YES (NO) column shows the number of systems determined to be (non-)ground-confluent together with average time (∅-time) the tool took. The ∞ column is the number of timeouts. To compare overall performance the *total time* column contains the sum of all run times, including timeouts but excluding the time taken by FORTify. The ✓ columns show the numbers of certifiable results as well as the overall time taken by FORTify. These results show that, even though they have the same meaning, the choice of formula has an impact on performance. Most notably this can be seen when comparing the number of solved problems by FORT-h 2.0. The formula (16) (semi-confluence) was fastest with no timeouts, followed by (15) with one timeout and (17) with two. It is apparent that formulas containing conversion (\leftrightarrow^*) are especially slow, which we will also see in later experiments. Further note that FORT-h 2.0 can solve an additional problem compared to the 1.0 version, for each formula.

Interestingly FORT-h (2.0) is generally faster and can solve more problems than FORT-j even though the latter implements parallelism. This performance advantage is more prominent in systems which are non-confluent where FORT-h can solve more problems, while for problems with the answer YES, FORT-j can solve close to the same number of problems, while taking less time per problem in general. The table also shows that FORTify can certify most of the results, which is a large improvement over the previous version. Here the difference between the three formulas is not as visible, but it is also faster on (16) and (15), and slowest on (17). The times for FORTify must also be seen in the context that it ran on more problems on the first two formulas, since FORT-h could produce more certificates. No wrong results by the decision tools were identified.

Experiment 2 The second set of formulas represents the normal form property, restricted to ground terms (GNFP):

$$\text{"forall } t, u (t \leftarrow^* u \ \& \ NF(u) \Rightarrow t \rightarrow^* u) \text{"} \tag{18}$$

$$\text{"forall } s, t, u (s \rightarrow t \ \& \ s \rightarrow ! u \Rightarrow t \rightarrow^* u) \text{"} \tag{19}$$

$$\text{"forall } t (WN(t) \Rightarrow CR(t)) \text{"} \tag{20}$$

The results for these are shown in Table 6. The same pattern is observed, where even though

Table 6 FORT-h (with FORTify) and FORT-j run on GNFP formulas

		YES	∅-time	✓	NO	∅-time	✓	∞	Total (✓) time	
(18)	FORT-h 2.0	59	0.30 s	57	63	0.04 s	63	0	20.37 s	(0.5h)
	FORT-h 1.0	59	0.70 s	31	63	0.07 s	20	0	45.62 s	(14.6h)
	FORT-j	59	0.23 s	–	63	0.39 s	–	0	38.16 s	
(19)	FORT-h 2.0	59	0.02 s	59	63	0.01 s	63	0	1.76 s	(0.1h)
	FORT-h 1.0	59	0.03 s	46	63	0.01 s	50	0	2.55 s	(6.3h)
	FORT-j	59	0.22 s	–	63	0.30 s	–	0	31.83 s	
(20)	FORT-h 2.0	59	0.03 s	56	62	0.11 s	62	1	68.83 s	(0.8h)
	FORT-h 1.0	59	0.05 s	42	62	0.12 s	45	1	70.51 s	(8.6h)
	FORT-j	59	0.31 s	–	62	0.64 s	–	1	117.86 s	

Table 7 FORT-h 2.0 run on " \sim forall s, t (s <->* t)" with differing encodings of conversion

	YES	∅-time	NO	∅-time	∞	total time
(21)	91	0.10 s	31	0.42 s	0	22.00 s
(22)	91	0.10 s	31	0.48 s	0	24.22 s
(23)	91	0.07 s	31	0.41 s	0	19.31 s

all three can (dis)prove satisfaction for the same formulas, FORT-h 2.0 is faster than FORT-j overall, and has improved over FORT-h 1.0.

Since the representations containing conversion (\leftrightarrow^*) in the previous experiments are outperformed by the other representations, it is often a good idea to avoid it. Obviously this is not always possible. Take the properties UNC, CE or consistency for example. It is therefore important to choose the correct representation in the primitive automata constructions, to ensure good performance when conversion cannot be avoided.

Experiment 3 We tested the following three representations of conversion for a TRS \mathcal{R} :

$$((\rightarrow_{\mathcal{R}}^{\varepsilon})^{-} \cup \rightarrow_{\mathcal{R}}^{\varepsilon})^{\hat{\top}} \supseteq \tag{21}$$

$$((\rightarrow_{\mathcal{R}}^{\varepsilon})^{-} \hat{\circ} \rightarrow_{\mathcal{R}}^{\varepsilon})^{\hat{\top}} \supseteq \tag{22}$$

$$((\rightarrow_{\mathcal{R} \cup \mathcal{R}^{-}}^{\varepsilon})^{\hat{\top}}) \supseteq \tag{23}$$

The representation (21) is the one listed in Table 2. Using composition ($\hat{\circ}$) instead of union as in (22) works because

$$(\rightarrow_{\mathcal{R}}^{\varepsilon})^{-} \hat{\circ} \rightarrow_{\mathcal{R}}^{\varepsilon} = ((\rightarrow_{\mathcal{R}}^{\varepsilon})^{-} \circ \twoheadrightarrow_{\mathcal{R}}) \cup ((\twoheadrightarrow_{\mathcal{R}})^{-} \circ \rightarrow_{\mathcal{R}}^{\varepsilon})$$

The third representation (23) uses the identity $\rightarrow_{\mathcal{R} \cup \mathcal{R}^{-}}^{\varepsilon} = \leftrightarrow_{\mathcal{R}}^{\varepsilon}$ and is the default used by FORT-h. The results of running FORT-h 2.0 on the COPS dataset, using the formula " \sim forall s, t (s <->* t)" for consistency with the three different representations of conversion can be seen in Table 7. We can see that (23) is the fastest with and overall runtime of 19.31 s. It is about 12% faster than (21) and about 20% faster than (22). Also important is that (23) produces smaller automata, which leads to better performance when conversion is embedded within larger formulas. Consider for example COPS #741:

$$\text{if}(\text{true}, a, x) \rightarrow a \qquad \text{if}(\text{true}, g(a), x) \rightarrow g(a) \qquad g(a) \rightarrow g(g(a))$$

Table 8 FORT-h 2.0 (with FORTify) run on normalization with different encodings of NF

	YES	∅-time	✓	NO	∅-time	✓	∞	total (✓) time
"NF (t) "	41	0.02s	41	81	0.00s	81	0	0.85s (20.50s)
"~exists u (t -> u) "	41	0.02s	41	81	0.00s	81	0	1.05s (23.71s)

$$\begin{array}{lll}
 \text{if}(\text{true}, b, x) \rightarrow b & \text{if}(\text{true}, g(b), x) \rightarrow g(b) & g(b) \rightarrow a \\
 \text{if}(\text{false}, x, a) \rightarrow a & \text{if}(\text{false}, x, g(a)) \rightarrow g(a) & f(a, b) \rightarrow b \\
 \text{if}(\text{false}, x, b) \rightarrow b & \text{if}(\text{false}, x, g(b)) \rightarrow g(b) & f(g(g(a)), x) \rightarrow b
 \end{array}$$

The RR_2 automata representing (21) and (22) both contain 233 states, 7927 transitions and 9 ϵ -transitions before trimming, and 132 states and 4937 transitions after. In comparison the automaton for (23) contains 152 states, 3975 transitions and 9 ϵ -transitions before, and 75 states with 2313 transitions after trimming. Overall (23) therefore has less than half the number of transitions in this example, which can have a significant effect in any later closure operations.

The final experiment in this subsection involves the normal form predicate $NF(t)$, which is implemented in FORT-h according to the description in Sect. 5.4, instead of using the equivalent formula $\neg \exists u (t \rightarrow u)$.

Experiment 4 Consider the formula "forall s (exists t (NF(t) & s ->* t))" for normalization and COPS #503:

$$f(a, a, b, b) \rightarrow f(c, c, c, c) \quad a \rightarrow b \quad a \rightarrow c \quad b \rightarrow a \quad b \rightarrow c$$

When using the formula $\neg \exists u (t \rightarrow u)$ for $NF(t)$, FORT-h first constructs the RR_2 automaton \mathcal{A}_1 for $t \rightarrow u$, with 4 states and 15 transitions. It then projects to construct the automaton \mathcal{A}_2 for $\exists u (t \rightarrow u)$ with 4 states and 13 transitions, and finally it has to determinize \mathcal{A}_2 and construct the complement for the negated formula $\neg \exists u (t \rightarrow u)$, resulting in the automaton \mathcal{A}_3 with 4 states and 259 transitions before and 1 state with two 2 transitions after trimming. If instead the direct normal form predicate is used, FORT-h immediately produces the latter automaton, without having to construct the intermediate automata or having to trim. The impact on runtime can be seen in Table 8. It is rather small for FORT-h, but for FORTify the direct construction is about 13% faster. When looking at the sizes of the automata, the average untrimmed automaton \mathcal{A}_3 , for our dataset of left-linear right-ground COPS problems, contains 75.8 transitions while the average automaton for the normal form predicate contains 13.3 transitions.

8.1.2 Properties Involving Multiple TRSs

We also ran experiments to test performance on properties involving two TRSs. As a dataset we constructed problems of all ordered pairs of COPS problems, resulting in 7503 pairs.

Experiment 5 The first property tested was ground-commutation (GCOM). The results, presented in Table 9, show that FORT-h is ahead of FORT-j here as well. It can (dis)prove more problems, timing-out on only two as compared to 49 problems. Additionally it does so in significantly less time. With FORTify we can see a large improvement over the old version. It is able to certify close to 98% of the results found by FORT-h 2.0.

Table 9 FORT-h (with FORTify) and FORT-j run on GCOM

	YES	\emptyset -time	✓	NO	\emptyset -time	✓	∞	total (✓) time
FORT-h 2.0	1381	0.10 s	1368	6120	0.02 s	5965	2	374.63 s (51.5h)
FORT-h 1.0	1381	0.16 s	878	6120	0.03 s	3666	2	517.32 s (681.5h)
FORT-j	1354	1.46 s	–	6100	0.94 s	–	49	10670.89 s

In the 2019 edition of the Confluence Competition [41] three tools contested the commutation (COM) category:⁸ ACP [2], CoLL [49], and FORT-j. On input problem COPS #1118 the tools gave conflicting answers.

Example 35 COPS #1118 is about the commutation of the TRSs COPS #669

$$a \rightarrow c \quad f(a) \rightarrow b \quad b \rightarrow b \quad b \rightarrow h(b, h(c, a))$$

and COPS #695

$$h(a, a) \rightarrow c \quad b \rightarrow h(b, a) \quad b \rightarrow a \quad f(c) \rightarrow c \quad c \rightarrow a$$

To determine the correct answer we use FORT-h 2.0 to produce a certificate for ground-commutation by calling

```
> fort-h -c cert -i "GCom([0],[1])" 1118.trs YES
```

This produces the following certificate:

```
(0 (rr2 (comp (inverse (step* (1))) (step* (0))) 0 1)
  (rr2 (comp (inverse (step* (1))) (step* (0))) 0 1)
  (size 13 53 0))
(1 (rr2 (comp (step* (0)) (inverse (step* (1)))) 0 1)
  (rr2 (comp (step* (0)) (inverse (step* (1)))) 0 1)
  (size 11 47 0))
(2 (not 1) (not (rr2 (comp (step* (0)) (inverse (step*
  (1)))) 0 1)))
(3 (and (0 2))
  (and ((rr2 (comp (inverse (step* (1))) (step* (0))) 0 1)
    (not (rr2 (comp (step* (0)) (inverse (step* (1))))
    0 1))))))
(4 (exists 3)
  (exists (and ((rr2 (comp (inverse (step* (1))) (step*
  (0))) 0 1) (not (rr2 (comp (step* (0)) (inverse (step*
  (1)))) 0 1))))))
(5 (exists 4)
  (exists (exists (and ((rr2 (comp (inverse (step* (1)))
  (step* (0))) 0 1) (not (rr2 (comp (step* (0))
  (inverse (step* (1)))) 0 1))))))
(6 (not 5)
  (not (exists (exists (and (
  (rr2 (comp (inverse (step* (1))) (step* (0))) 0 1)
```

⁸ <https://cops.uibk.ac.at/results/?y=2019&c=COM>

Table 10 FORT-h 2.0 (with FORTify) run on (G)CE and G(NE)

	YES	∅-time	✓	NO	∅-time	✓	∞	total (✓) time	
GCE	157	0.70s	150	7162	0.94s	6736	184	5.0h	(125.6h)
CE	151	0.74s	144	7168	0.93s	6739	184	5.0h	(127.1h)
GNE	181	0.02s	181	7320	0.04s	7308	2	448.75s	(5.4h)
NE	177	0.02s	177	7324	0.04s	7312	2	446.54s	(5.6h)

```

(not (rr2 (comp (step* (0)) (inverse (step* (1))))
 0 1))))))
(7 (nmf 6)
 (forall (forall (or (
 (not (rr2 (comp (inverse (step* (1))) (step* (0))
 )) 0 1)) (rr2 (comp (step* (0)) (inverse (step* (1))
 s))0 1))))))
(nonempty 7)

```

When passing this certificate to FORTify, after 0.2s the output *Certified* is produced, so we can be assured that the TRSs do commute. Note that the inference steps 0 and 1 contain the optional size information. Here $(size\ k\ m\ n)$ means the underlying RR_n automaton constructed by FORT-h 2.0 contains k states, m transitions, and n ϵ -transitions.

Experiment 6 For the second experiment using multiple TRSs we tested FORT-h 2.0 and FORTify on conversion equivalence and normalization equivalence, once for all terms and once for only ground-terms. FORT-h 1.0 and FORT-j have not implemented the necessary signature extension results to cover these properties, and are therefore not run. The results can be seen in Table 10. Comparing the properties to the corresponding ground-properties, we can see that FORT-h 2.0 succeeds to find results on the same number of problems. However, six results moved from YES to NO in the case of (G)CE and four in the case of (G)NE. These correspond to TRSs where the additional constants are needed to disprove the property. While the run times of FORT-h 2.0 stayed almost the same when comparing the ground and non-ground properties, we can see that FORTify does take longer to certify results on the non-ground properties. This is to be expected, since the additional constants lead to larger automata. Simply by having a larger signature, some of the atomic constructions produce more transition rules. While this is usually only a small difference it can have a significant effect when embedded within a bigger formula.

8.1.3 Optimizations

To show this effect, and the improvement caused by Lemma 39 consider the following example.

Example 36 Consider COPS #214

$$a \rightarrow b \quad a \rightarrow f(a) \quad b \rightarrow f(f(b)) \quad f^{64}(b) \rightarrow b$$

where f^{64} represents 64 nested applications of f . To check UNC, FORT-h 2.0 extends the signature as needed and uses the formula for GUNC internally represented as

$$\neg \exists (\exists ((NF(0) \times NF(1)) \wedge 0 \neq 1 \wedge 0 \leftrightarrow^* 1))$$

Table 11 FORT-h 2.0 run on UNC with and without Lemma 39

		YES	∅-time	NO	∅-time	∞	total time
"UNC"	(with Lemma 39)	72	0.29 s	49	0.20 s	1	90.92 s
"{+2} GUNC"	(two constants)	72	0.54 s	49	0.20 s	1	108.52 s

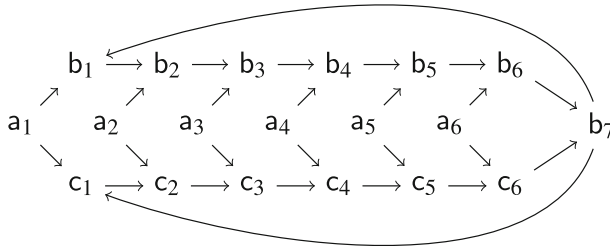


Fig. 12 Graph presentation of COPS #116

In this case no constants have to be added, since the TRS is ground. The intermediate automaton \mathcal{A}_1 for the subformula $NF(0) \times NF(1)$ contains no transitions, since the TRS has no normal forms for the given signature. For the automaton \mathcal{A}_2 of $0 \neq 1$ we have 13 transitions and \mathcal{A}_3 for $0 \leftrightarrow^* 1$ has 150,569 transitions. Like we have seen in earlier experiments, the automaton for conversion is clearly the largest, and would also take the largest amount of time to construct. However, since \mathcal{A}_1 is empty, the intersection with \mathcal{A}_2 and then further with \mathcal{A}_3 will also be empty. And due to the lazy evaluation strategy of Haskell the third automaton will never be constructed. Therefore FORT-h 2.0 can almost instantly (0.01 s) determine that the automaton for the formula within the negation is empty, and conclude that UNC holds. However, if we were to ignore the optimization introduced by Lemma 39 and add two constants the automaton \mathcal{A}_1 is no longer empty, since we added two normal forms to the domain. This changes the numbers as follows: The automaton \mathcal{A}_1 would contain 15 transitions and 3 states, \mathcal{A}_2 has 31 transitions and 3 states, and \mathcal{A}_3 has 150,571 transitions and 4356 states. Since none of the automata are empty we must construct the intersection $\mathcal{A}_1 \cap \mathcal{A}_2$ containing 34 transitions and 6 states. After trimming this drops to 20 transitions and 4 states. The intersection $(\mathcal{A}_1 \cap \mathcal{A}_2) \cap \mathcal{A}_3$ then results in an automaton with 132,652 transitions and 8584 states. Only after trimming we see that this automaton is empty to conclude that UNC holds. Overall this takes FORT-h 2.0 7.15 s, which is orders of magnitude slower than with the optimization. While such large speedups are not the norm, the overall runtime on the COPS dataset for UNC drops by about 16%, as seen in Table 11.

Example 37 To see that the optimization of collapsing strongly connected states, introduced in Sect. 7.1, can have a significant effect consider COPS #116. It is an ARS consisting of 26 rules presented as a graph in Fig. 12. To check if it is consistent we can use the formula " \sim forall s, t (s \leftrightarrow^* t)" which is internally represented as $\exists(\exists(\neg(0 \leftrightarrow^* 1)))$. For this FORT-h constructs the automaton \mathcal{A} for $0 \leftrightarrow^* 1$, consisting of 8 states 418 transitions and 3 ε -transitions. After eliminating the ε -transitions and trimming, we are left with 1 state and 361 transitions. The complement automaton \mathcal{A}^c which represents $\neg(0 \leftrightarrow^* 1)$ has the same size, which drops to zero after trimming, showing that the system is not consistent. Overall FORT-h takes 0.34 s.

If we however remove the optimization and do not collapse strongly connected components, we get significantly larger automata. The automaton \mathcal{A} grows to 8427 states, 2827

Table 12 FORT-h 2.0 run on " \sim forall s, t (s \leftrightarrow * t)" with/out collapsing SCCs

	YES	\emptyset -time	NO	\emptyset -time	∞	Total time
Collapsing SCCs	91	0.07 s	31	0.41 s	0	19.31 s
Unoptimized	91	0.14 s	28	1.21 s	3	223.82 s

Table 13 FORT-h 2.0 compared to other tools

		YES	\emptyset -time	NO	\emptyset -time	∞ /MAYBE	Total time
GCR	FORT-h 2.0	37	0.06 s	84	0.04 s	1	65.82 s
	AGCP	24	0.02 s	79	0.07 s	19	276.42 s
NFP	FORT-h 2.0	55	0.02 s	67	0.01 s	0	1.76 s
	CSI	55	0.79 s	61	1.02 s	6	186.94 s
UNC	FORT-h 2.0	72	0.31 s	49	0.21 s	1	92.75 s
	ACP	70	0.08 s	47	0.86 s	5	345.91 s
	CSI	71	0.83 s	46	1.12 s	5	187.37 s
UNR	FORT-h 2.0	96	0.02 s	26	0.01 s	0	2.21 s
	CSI	86	0.81 s	26	0.76 s	10	209.12 s
COM	FORT-h 2.0	1365	0.10 s	6135	0.04 s	3	578.3 s
	CoLL	1349	0.21 s	4015	0.13 s	2139	19.5 h
	ACP	1238	0.01 s	3519	0.04 s	2746	5.0 h

transitions and 851,916 ε -transitions. At this point the procedure usually eliminates the ε -transitions and trims the automaton, but FORT-h does not manage to do so within the 60 s timeout. The overall improvement on testing consistency can be seen in Table 12.

8.1.4 Comparison with Other Tools

As a last experiment we compare FORT-h to a number of state of the art tools. For the properties GCR, NFP, UNC, UNR and COM we chose the following tools that competed in the corresponding categories in the confluence competition in 2021: ACP [2] in UNC and COM, AGCP [1] in GCR, CSI [44] in NFP, UNC and UNR, and CoLL [49] in COM. All these tools implement various sufficient conditions for the corresponding property and are not limited to linear variable-separated or left-linear right-ground TRSs. For the sake of comparing them to FORT-h we run them only on the left-linear right-ground TRSs in COPS, and on the pairs of these problems for COM. The results can be seen in Table 13.

We can see that FORT-h 2.0 significantly outperforms all the other tools on this class of systems. For all properties it can find results for more problems and can often do so with less time per problem. This difference is especially pronounced in the COM category, where FORT-h 2.0 can (dis)prove all but three of the 7503 problems, while ACP and CoLL timeout or return Maybe on more than 2000 of these. Given this performance discrepancy it is of interest to other tools to use FORT-h 2.0 on problems of this class. Here it could be used as a black box on problems (or subproblems) as long as they are linear variable-separated

TRSs, and can be expressed in the first-order theory of rewriting. An example of such a tool is CONFIDENT [27] which uses FORT, among other tools, as part of its procedure.⁹

Another interesting point can be seen when comparing the first line in Table 13, where 37 YES results are reported, with the fourth line in Table 5, where 38 YES results are reported. Both formulas check ground-confluence, but the built-in GCR property is represented slightly different. Instead of the joinability predicate $(t \downarrow u)$, which is constructed via operations on anchored GTTs, it uses the equivalent formula $\exists v (t \rightarrow^* v \wedge u \rightarrow^* v)$. In this case the explicit formula is slower on COPS #215 leading to the additional timeout, but is faster on other problems causing the total time to be similar. Like previous experiments this shows that the representation of a property can have a large and non-obvious effect on performance.

8.2 FORT-s

In this subsection we report on the synthesis experiments that we performed. All experiments were executed with the options `-j 4` and `+RTS -A64M`, unless stated otherwise. First we consider Fig. 6.

Experiment 7 The following TRSs were produced by FORT-s on the given formulas when restricting the signature (using the command-line option `-S "a 0 b 0 f 2"`) to a binary function symbol `f` and two constants `a` and `b`:

"GWCR & ~WCR & ~GCR"	$a \rightarrow b$	$f(a, x) \rightarrow a$	$a \rightarrow f(a, a)$	9 s
"GCR & ~CR & ~GSCR"	$a \rightarrow b$	$f(a, x) \rightarrow f(a, a)$	$f(b, b) \rightarrow a$	10 s
"GNFP & ~NFP & ~GCR"	$a \rightarrow b$	$f(a, x) \rightarrow f(a, a)$	$f(b, b) \rightarrow f(a, a)$	4 s
"GUNC & ~UNC & ~GNFP"	$a \rightarrow a$	$f(a, x) \rightarrow a$	$f(x, b) \rightarrow b$	11 s

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

$$\begin{array}{llllll}
 b \rightarrow a & c \rightarrow c & d \rightarrow c & f(x, a) \rightarrow A & f(x, A) \rightarrow A \\
 b \rightarrow c & & d \rightarrow e & f(x, e) \rightarrow A & f(c, x) \rightarrow A
 \end{array}$$

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

In the next experiment we use the infinity predicate to distinguish well-known subclasses of linear-variable separated TRSs.

Experiment 8 The formula

$$\exists t \text{ INF}_{\leftarrow}^{\varepsilon}(t) \qquad \text{"exists } t \text{ (INF(e<- , t))"}$$

distinguishes ground TRSs from left-linear right-ground (but not ground) ones. Without any options FORT-s produces the TRS $\{g(x) \rightarrow g(a)\}$ in a fraction of a second. The formula

$$\exists t \text{ INF}_{\neq}(t) \qquad \text{"exists } t \text{ (INF(~ = , t))"}$$

⁹ <http://zenon.dsic.upv.es/confident>

is true for TRSs that are not ARSs. FORT-s produces the empty TRS over the signature consisting of the constant *a* and an additional constant and unary function symbol. The second constant is not necessary, but is added by the signature step. Finally, to distinguish linear variable-separated TRSs from left-linear right-ground TRSs, assuming the signature contains at least one non-constant function symbol, the formula

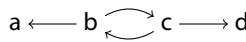
$$\exists t \text{ INF}_{\rightarrow} (t) \quad \text{"exists } t \text{ (INF}(-\rightarrow e, t)\text{) "}$$

can be used in connection with the `-l` option. This generates the TRS $\{a \rightarrow x\}$ over the signature consisting of the constant *a* and an additional constant and unary function symbol. Without the latter, the generated linear variable-separated TRS induces only a finite rewrite relation. Adding `"& CR & WN"` to the last formula produces the TRS $\{a \rightarrow b, f(b) \rightarrow x\}$.

Experiment 9 Finding a locally confluent but not confluent TRS \mathcal{R} is easy. FORT-s produces the ground TRS

$$a \rightarrow b \qquad f(a) \rightarrow a \qquad a \rightarrow f(a)$$

when given the formula `"WCR & ~CR"` is less than 1s. The well-known abstract counterexample by Kleene



is found by restricting the search to ARSs. The easiest way to do this is with the option `-A 0`, which sets the maximal arity of function symbols to 0. Moreover, the maximum number of rewrite rules has to be set to at least four (`-R 4`). If we impose the additional condition that \mathcal{R}^- is terminating (cf. [56]), the TRS

$$a \rightarrow b \qquad a \rightarrow g(a) \qquad b \rightarrow g(g(b))$$

is generated with

$$\text{"WCR & ~CR & ~exists } t \text{ (INF}(*\leftarrow-, t) \mid t \leftarrow- t)\text{"}$$

without any additional command-line options in less than 7s.

The next experiment shows how FORT-s can be used to complete TRSs into complete (canonical) ones.

Experiment 10 FORT-s produces the TRS $\{a \rightarrow c, f(x) \rightarrow a\}$ when presented the formula `"[0] (WCR & SN) & forall s, t ([0] s <->* t <=> [1] s <->* t)"` with `input.trs` as additional parameter. Here `input.trs` consists of the three rules

$$c \rightarrow a \qquad f(b) \rightarrow c \qquad f(c) \rightarrow a$$

The result is complete (as demanded by `"[0] (WCR & SN)"`), but not equivalent! The reason is that `"forall s, t ([0] s <->* t <=> [1] s <->* t)"` ensures ground conversion equivalence, and we have seen in Sect. 6 that an extra constant is needed to reduce conversion equivalence to ground conversion equivalence. The same behaviour can also be seen for our leading example, where the same formula is used. When presented the formula

$$\text{" [0] (WCR & SN) & CE([0], [1]) "}$$

the equivalent complete TRS consisting of the rules

$$a \rightarrow c \qquad f(b) \rightarrow f(a) \qquad f(c) \rightarrow a$$

is synthesized. Note that the latter TRS is not canonical since not all right-hand sides are in normal form. It is well-known that every system of ground equations admits a presentation as canonical TRS. Snyder [50] proved that a ground TRS is canonical if and only if it is *reduced*. The latter property is easily expressible:

```
"[0] (forall s, t (s ->e t => NF(t) & ~exists u (s ->be u) &
  forall u (s ->e u => t = u)))"
```

Together with "CE ([0], [1])", any ground TRS is transformed into an equivalent canonical one, without explicitly requiring confluence and termination. For our example TRS, we obtain

$$a \rightarrow c \qquad f(b) \rightarrow c \qquad f(c) \rightarrow c$$

The final experiment is based on [57, Example 5.1] and shows how FORT-s can be used to synthesize multiple TRSs.

Experiment 11 If we want to generate two terminating ARSs such that their union is non-terminating, the formula "[0] SN & [1] SN & ~SN" can be used in connection with the options `-A 0` and `-n 2`. The latter tells FORT-s to synthesize two TRSs. 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)"
```

In a fraction of a second FORT-s synthesizes the following two ARSs satisfying the conjunction of these requirements:

$$\mathcal{A}_0 : a \rightarrow b \quad b \rightarrow c \qquad \mathcal{A}_1 : b \rightarrow c \quad c \rightarrow a$$

Using completely different techniques, similar ARSs are generated by Carpa, the tool described in Zantema [57].

9 Conclusion

In this paper we presented a formalized decision procedure of the first-order theory of rewriting for the class of linear variable-separated TRSs. The decision procedure ultimately goes back to Dauchet and Tison [10] and is the basis of the tool FORT-h. Different from [8, 10], we extensively use *anchored* GTT relations. These have better closure properties than GTT relations and allow to efficiently express numerous binary relations on ground terms, easing formalization efforts. We presented signature extension results that allow us to reduce certain properties on arbitrary terms to the corresponding properties on ground terms. These allow FORT-h to participate in categories other than GCR in the Confluence Competition. We presented a certificate language in which certificates for the yes/no output of the decision procedure can be expressed. These certificates are validated by FORTify, the verified Haskell program obtained from the executable Isabelle formalization. FORT-h supports properties like commutation that involve multiple TRSs. Witness generation is useful to gain insight in

why a particular property holds. The synthesis mode is used to find small TRSs that satisfy a given property. FORT-s supports several options to control the (infinite) search space. We showed that the synthesis problem is undecidable, already for ARSs, by a reduction from PCP.

Comprehensive experimental results were presented, including a comparison with the tools ACP [2], AGCP [1], CoLL [49], CSI [44] that compete with FORT-h in CoCo. Full details are available from the web site <https://fortissimo.uibk.ac.at/> which additionally provides a convenient interface to FORT-h, FORT-s and FORTify, as well as precompiled binaries for the three tools.

Linear variable-separated TRSs are a proper extension of left-linear right-ground TRSs. Dropping either restriction, one quickly faces an undecidable first-order theory, even when one-step rewriting (\rightarrow) is the only predicate. This was first shown by Treinen [54]. Related undecidability results are presented in [39, 55]. In particular, Marcinkowski [39] showed that the first-order theory of one-step rewriting is undecidable for right-ground TRSs.

Many concrete properties expressible in the first-order theory of rewriting are known to be decidable for much larger classes of rewrite systems. For instance, termination is known to be decidable for right-linear right-shallow TRSs, a result by Godoy et al. [25], extending the earlier decision result for right-ground systems of Dershowitz [14]. Termination is also decidable for almost-orthogonal growing TRSs [43]. Confluence is decidable for right-linear shallow TRSs [24] and for right-ground TRSs [30].

For ground TRSs, which are in the scope of FORT-h, termination is known to be decidable in polynomial time [45]. The same holds for confluence [7]. Felgenhauer [19] showed that confluence can be decided in cubic time. Similar complexity results for the related properties NFP, UNC and UNR are given in [20]. The worst-case complexity of the formalized decision procedure implemented in FORT-h is at least double exponential (cf. [26]).

Concerning synthesis, 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 [29] developed a synthesis tool to attack the well-known open problems [15, 16] concerning the sufficiency of certain restricted joinability conditions on critical pairs of left-linear TRSs. Zantema [56] 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 shape $a \rightarrow b$, $a \rightarrow f(b)$, or $f(a) \rightarrow b$, 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 [57] synthesized combinations of ARSs with help of a SAT solver. It was used to show the necessity of certain conditions in abstract confluence results [52, Sect. 5] and inspired us to support multiple TRSs in FORT.

Concerning future work, improving the efficiency of FORT-h by supporting parallelism might result in a speed-up, especially for larger formulas. The minimization of tree automata (also non-deterministic ones) is an obvious target for further investigation. Preprocessing techniques that go beyond the mere transformation to negation normal form will be helpful to obtain equivalent formulas that reduce the size of the ensuing tree automata in the decision procedure. In [28] similar ideas are applied to WSkS, in connection with MONA [31]. An interesting question is whether FORT-h can be extended to deal with properties involving innermost and other restrictions of rewriting. Formalization efforts that aim to transfer code in module A to the verified code in module B in Fig. 7, are also of interest. The conversion of FORT syntax to de Bruijn notation is a natural candidate here.

Acknowledgements This research was supported by FWF (Austrian Science Fund) project P30301. Several persons helped to make this project successful. We are grateful to Bertram Felgenhauer for numerous contributions. Franziska Rapp implemented the first versions of FORT in OCaml and Java. She and T. V. H. Prathamesh contributed to the early stage of the formalization of the decision procedure. Jamie Hochrainer reimplemented the synthesis mode, resulting in FORT-s. Johannes Koch designed the web interface. We thank René Thiemann for advice concerning turning the formalization into executable code. The first author acknowledges the support of the Future Value Creation Research Center of Nagoya University, where part of the research was performed. The detailed comments of the anonymous reviewers improved the presentation.

Author Contributions All authors contributed to the research reported in the manuscript. Alexander Lochmann performed the formalizations in Isabelle/HOL that led to FORTify. Fabian Mitterwallner was the main developer of the artifacts (FORT-h, FORT-s and FORTify). The first draft of the manuscript was written by Aart Middeldorp and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding Open access funding provided by Austrian Science Fund (FWF). This work was supported by FWF (Austrian Science Fund) project P30301. The first author acknowledges the support of the Future Value Creation Research Center of Nagoya University.

Data Availability The experiments summarized in the manuscript are available from <https://fortissimo.uibk.ac.at/jar>. The same holds for binaries and sources of the artifacts.

Declarations

Conflict of interest The author declares that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Appendix A: Input Format

The input format of FORT-h can be roughly split into two parts: The logical structure of the property and the involved atomic predicates and relations. The logical structure is defined by the following grammar, where angle brackets $\langle \rangle$ are used for non-terminal symbols:

$$\begin{aligned}
 \langle formula \rangle & ::= \langle formula \rangle \langle operator \rangle \langle formula \rangle \mid \sim \langle formula \rangle \\
 & \quad \mid \langle quantifier \rangle \langle vars \rangle (\langle formula \rangle) \mid \langle var \rangle \langle relation \rangle \langle var \rangle \\
 & \quad \mid \langle property \rangle \mid \{ + \langle nat \rangle \} \langle formula \rangle \mid [\langle trss \rangle] \langle formula \rangle \\
 & \quad \mid (\langle formula \rangle) \\
 \langle operator \rangle & ::= \langle = \rangle \mid \langle = \rangle \mid \langle = \rangle \mid \langle = \rangle \mid \langle = \rangle \\
 \langle quantifier \rangle & ::= \text{forall} \mid \text{exists} \\
 \langle trss \rangle & ::= \langle nat \rangle \mid \langle nat \rangle, \langle trss \rangle \\
 \langle vars \rangle & ::= \langle var \rangle \mid \langle var \rangle, \langle vars \rangle
 \end{aligned}$$

Here $\langle nat \rangle$ is a natural number, $\langle var \rangle$ is an alphanumeric string representing a variable name and $\langle trss \rangle$ is a comma separated list of indices referencing TRSs. The logical operators

are all right-associative. Regarding precedence the unary operations bind strongest with the binary operators respecting the order $\> | \> \Rightarrow \> \Leftarrow$. Most represented operations have the meaning expected from a first-order formula, the exception being the operations $\{+\langle nat \rangle\} \langle formula \rangle$, which allows the user to specify the number of constants to be added to the signature when evaluating the subformula, and $[\langle trss \rangle] \langle formula \rangle$, which restricts and permutes the indices of TRSs for the underlying subformula.

The atomic binary relations supported by FORT-h are defined as:

$$\begin{aligned} \langle relation \rangle ::= & \rightarrow_e | \rightarrow_e^* | \rightarrow_e = | \rightarrow_e + | e \leftarrow | *e \leftarrow | =e \leftarrow | +e \leftarrow \\ & | \rightarrow_{be} | \rightarrow_{be}^* | \rightarrow_{be} = | \rightarrow_{be} + | be \leftarrow | *be \leftarrow | =be \leftarrow | +be \leftarrow \\ & | \rightarrow | \rightarrow^* | \rightarrow = | \rightarrow + | \leftarrow | * \leftarrow | = \leftarrow | + \leftarrow \\ & | \rightarrow ! | - | | \rightarrow | ! \leftarrow | \leftarrow | | - | \leftarrow \rightarrow | \leftarrow \rightarrow^* \\ & | = | join | meet \end{aligned}$$

Here the \rightarrow_e stands for a root step, \rightarrow_{be} for a step below the root, \rightarrow a normal rewrite step, $\rightarrow!$ is a reduction to normal form, $- | \rightarrow$ is a parallel step, $join$ stands for joinability \downarrow and $meet$ for meetability \uparrow . The suffix $*$ stands for the transitive-reflexive, $+$ for the transitive, and $=$ for the reflexive closures.

Example 38 Consider calling FORT-h with three input TRSs on the formula:

$$"+2\} forall s, t ([2,0] ([0] s \rightarrow ! t \Leftarrow [1] s \rightarrow ! t))"$$

The $\{+2\}$ instructs FORT-h to add two constants to the signature when constructing the automata. Normally " $[0] s \rightarrow ! t$ " means that term s normalizes to term t in the first input TRS (the one with index 0), however here the context has changed due to the restrict modifier $[2, 0]$, which permutes and restricts the three TRSs in the subformula $([0] s \rightarrow ! t \Leftarrow [1] s \rightarrow ! t)$ such that $[0]$ refers to the TRS with index 2 and $[1]$ refers to the TRS with index 0. So FORT-h checks normalization equivalence of the third and first input TRS, while ignoring the second one. The two constants are added according to Table 3, since one of the involved TRSs may be linear variable-separated.

It is also possible to use some predefined properties by name. Here we differentiate between properties of terms and properties of whole TRSs.

$$\langle property \rangle ::= \langle prop_of_term \rangle | \langle prop_of_system \rangle$$

The properties on whole TRSs have the same names as defined in Sect. 6.

$$\begin{aligned} \langle prop_of_system \rangle ::= & CR | WCR | SCR | NFP | UNC | UNR | WN | SN \\ & | GCR | GWCR | GSCR | GNFP | GUNC | GUNR \\ & | \langle binary_prop \rangle ([\langle trss \rangle], [\langle trss \rangle]) \\ \langle binary_prop \rangle ::= & COM | GCOM | CE | GCE | NE | GNE \end{aligned}$$

The term properties take a variable as an additional argument.

$$\begin{aligned} \langle prop_of_term \rangle ::= & \langle prop \rangle (\langle var \rangle) | \langle finiteness \rangle (\langle binrel \rangle, \langle var \rangle) \\ \langle prop \rangle ::= & CR | WCR | WN | NFP | SN | NF | SCR | UNR \\ \langle finiteness \rangle ::= & INF | FIN \\ \langle binrel \rangle ::= & \langle binrel \rangle \langle operator \rangle \langle binrel \rangle | \sim \langle binrel \rangle | \langle relation \rangle \end{aligned}$$

Note that the INF and FIN properties also take a binary relation as an argument. This is usually one of the predefined rewrite relations, but may also be a more complex relation constructed by combining the rewrite relations using logical operators.

The property names (with exception of NF and INF) are all just a shorthand for larger formulas. In general these correspond to the definitions of the property in Sect. 6. However there are some exceptions. Take for example ground-confluence (GCR). This unfolds to the formula

$$\text{forall } s, t, u \ (s \rightarrow u \ \& \ s \rightarrow^* t \Rightarrow \text{exists } v \ (u \rightarrow^* v \ \& \ t \rightarrow^* v))$$

The $s \rightarrow u$ on the left of the implication differs from the original definition of GCR . However this property (known as semi-confluence [3]) can be shown to be equivalent to GCR by a simple induction proof, and generally leads to smaller automata in the decision procedure. The runtime comparison between different representations of ground-confluence and other properties is shown in Sect. 8.

Appendix B: User Interface of FORT-h

The command-line interface of FORT-h is

```
fort-h [OPTIONS] FORMULA TRS.trrs ..
```

where `TRS.trrs ..` is one or more files containing TRSs in the COPS format used in CoCo. It also supports *many-sorted* TRSs in the MSTRS format in the GCR category. The additional options are

- c FILE write certificate to FILE
- i enable the additional *<info>* in the inference steps of the certificate
- v enables verbose output (e.g., the internal representation)
- w enables witness generation

Witness generation enables the tool to produce witnesses/counterexamples and will be described in detail later in this section. For now, consider Example 28 and the call

```
> fort-h -w "CR" input.trrs
NO
formula body / witness:
  (0 (<- o ->*) 1 & ~ 0 (->* o *<-) 1)
  0 = g(_00())
  1 = g(_01())
```

So in addition to the answer NO, it also outputs a counterexample for the given formula consisting of the two terms $g_00()$ and $g_01()$. Here $_00$ and $_01$ are additional constants required to reduce confluence to ground-confluence, and represent variables. The terms should therefore be read as $g(x)$ and $g(y)$.

Appendix C: User Interface of FORT-s

The command-line interface of FORT-s is given below:


```
fort-s [OPTIONS] FORMULA [TRS.trrs ..]
```

where [TRS.trrs..] are zero or more files containing TRSs, and the options are

```
-j NUM      jobs to run in parallel (default: 1)
-l          search for linear variable-separated TRSs
-n NUM      number of systems to be synthesized (default: 1)
-S STRING   specifies signature (default: uses signature step)
-a STRING   specifies arities (default: uses signature step)
-s NUM      signature step (default: 2)
-A NUM      maximal generated arity (default: 3)
-D NUM      upper bound on height (default: 3)
-r NUM      lower bound on number of rules per system (default: 0)
-R NUM      upper bound on number of rules per system (default: 3)
-v NUM      upper bound on number of variables (default: 1)
```

The signature used during synthesis can be specified in multiple ways, the two simplest being with the command line flags `-S` and `-a`. With the option `-S` the signature is specified by a string listing the symbols in \mathcal{F} together with their arities, like in the call

```
fort-s -S "a 0 f 2 g 1" "GCR & ~CR"
```

Since we often do not care about the presentation of function symbols it is also permitted to just list arities with the option `-a`:

```
fort-s -a "0 1" "WN & ~SN"
```

FORT-s then generates unique symbol names for the user. If no signature is given, FORT-s generates successive signatures in a systematic manner with the help of a *signature step* and a bound on the maximal arity. If the signature step number is set to 1 and the arity is bounded by 3, signatures with the following arities are created:

$$\{0\}, \{0, 1\}, \{0, 1, 2\}, \{0, 1, 2, 3\}, \{0, 0, 1, 2, 3\}, \{0, 0, 1, 1, 2, 3\}, \dots$$

If the signature step is set to 2 (its default value), we obtain

$$\{0\}, \{0, 0\}, \{0, 0, 1\}, \{0, 0, 1, 1\}, \{0, 0, 1, 1, 2\}, \dots, \\ \{0, 0, 1, 1, 2, 2, 3, 3\}, \{0, 0, 0, 1, 1, 2, 2, 3, 3\}, \dots$$

The signature step is passed to FORT-s with the option `-s` and the bound on the arities by `-A`. Note that when additional systems are passed to FORT-s, it will use the union of the signatures of those systems.

When synthesizing n TRSs, in the given formula the indices 0 through $n - 1$ refer to the systems to be generated, and the indices greater than $n - 1$ refer to systems passed as additional inputs to FORT-s.

References

1. Aoto, T., Toyama, Y.: Ground confluence prover based on rewriting induction. In: Kesner, D., Pientka, B. (eds.) Proc. 1st International Conference on Formal Structures for Computation and Deduction. Leibniz International Proceedings in Informatics, vol. 52, pp. 33:1–33:12 (2016). <https://doi.org/10.4230/LIPIcs.FSCD.2016.33>
2. Aoto, T., Yoshida, J., Toyama, Y.: Proving confluence of term rewriting systems automatically. In: Treinen, R. (ed.) Proc. 20th International Conference on Rewriting Techniques and Applications. Lecture Notes in Computer Science, vol. 5595, pp. 93–102 (2009). https://doi.org/10.1007/978-3-642-02348-4_7

3. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998). <https://doi.org/10.1017/CBO9781139172752>
4. Berghofer, S.: First-order logic according to Fitting. Archive of Formal Proofs (2007). <https://isa-afp.org/entries/FOL-Fitting.html>
5. Berghofer, S., Bulwahn, L., Haftmann, F.: Turning inductive into equational specifications. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Proc. 22nd International Conference on Theorem Proving in Higher Order Logics. Lecture Notes in Computer Science, vol. 5674, pp. 131–146 (2009). https://doi.org/10.1007/978-3-642-03359-9_11
6. Comon, H.: Sequentiality, monadic second-order logic and tree automata. Inf. Comput. **157**(1–2), 25–51 (2000). <https://doi.org/10.1006/inco.1999.2838>
7. Comon, H., Godoy, G., Nieuwenhuis, R.: The confluence of ground term rewrite systems is decidable in polynomial time. In: Proc. 42th IEEE Symposium on Foundations of Computer Science, pp. 298–307 (2001). <https://doi.org/10.1109/SFCS.2001.959904>
8. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2008). <http://tata.gforge.inria.fr/>
9. Dauchet, M., Tison, S.: Decidability of confluence for ground term rewriting systems. In: Budach, L. (ed.) Proc. 5th International Conference on Fundamentals of Computation Theory. Lecture Notes in Computer Science, vol. 199, pp. 80–84 (1985). <https://doi.org/10.1007/BFb0028794>
10. Dauchet, M., Tison, S.: The theory of ground rewrite systems is decidable. In: Proc. 5th IEEE Symposium on Logic in Computer Science, pp. 242–248 (1990a). <https://doi.org/10.1109/LICS.1990.113750>
11. Dauchet, M., Tison, S.: The theory of ground rewrite systems is decidable (extended version). Technical Report I.T. 197, LIFL (1990b)
12. Dauchet, M., Heuillard, T., Lescanne, P., Tison, S.: Decidability of the confluence of finite ground term rewriting systems and of other related term rewriting systems. Inf. Comput. **88**(2), 187–201 (1990). [https://doi.org/10.1016/0890-5401\(90\)90015-A](https://doi.org/10.1016/0890-5401(90)90015-A)
13. de Bruijn, N.G.: Lambda calculus notation with nameless dummies: A tool for automatic formula manipulation, with application to the Church-Rosser theorem. Indagationes Mathematicae **34**(5), 381–392 (1972). [https://doi.org/10.1016/1385-7258\(72\)90034-0](https://doi.org/10.1016/1385-7258(72)90034-0)
14. Dershowitz, N.: Termination of linear rewriting systems (preliminary version). In: Even, S., Kariv, O. (eds.) Proc. 8th International Colloquium on Automata, Languages and Programming, vol. 115, pp. 448–458 (1981). https://doi.org/10.1007/3-540-10843-2_36
15. Dershowitz, N.: Open. Closed. Open. In: Giesl, J. (ed.) Proc. 16th International Conference on Rewriting Techniques and Applications. Lecture Notes in Computer Science, vol. 3467, pp. 276–393 (2005). https://doi.org/10.1007/978-3-540-32033-3_28
16. Dershowitz, N., Jouannaud, J.-P., Klop, J.W.: Open problems in rewriting. In: Book, R.V. (ed.) Proc. 4th International Conference on Rewriting Techniques and Applications. Lecture Notes in Computer Science, vol. 488, pp. 445–456 (1991). https://doi.org/10.1007/3-540-53904-2_120
17. Deruyver, A., Gilleron, R.: The reachability problem for ground TRS and some extensions. In: Proc. 14th Colloquium on Trees in Algebra and Programming. Lecture Notes in Computer Science, vol. 351, pp. 227–243 (1989). https://doi.org/10.1007/3-540-50939-9_135
18. Durand, I., Middeldorp, A.: Decidable call-by-need computations in term rewriting. Inf. Comput. **196**(2), 95–126 (2005). <https://doi.org/10.1016/j.ic.2004.10.003>
19. Felgenhauer, B.: Deciding confluence of ground term rewrite systems in cubic time. In: Tiwari, A. (ed.) Proc. 23rd International Conference on Rewriting Techniques and Applications. Leibniz International Proceedings in Informatics, vol. 15, pp. 165–175 (2012). <https://doi.org/10.4230/LIPIcs.RTA.2012.165>
20. Felgenhauer, B.: Deciding confluence and normal form properties of ground term rewrite systems efficiently. Log. Methods Comput. Sci. (2018). [https://doi.org/10.23638/LMCS-14\(4:7\)2018](https://doi.org/10.23638/LMCS-14(4:7)2018)
21. Felgenhauer, B., Thiemann, R.: Reachability, confluence, and termination analysis with state-compatible automata. Inf. Comput. **253**(3), 467–483 (2017). <https://doi.org/10.1016/j.ic.2016.06.011>
22. Felgenhauer, B., Middeldorp, A., Prathamesh, T.V.H., Rapp, F.: A verified ground confluence tool for linear variable-separated rewrite systems in Isabelle/HOL. In: Mahboubi, A., Myreen, M.O. (eds.) Proc. 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, pp. 132–143 (2019). <https://doi.org/10.1145/3293880.3294098>
23. Giesl, J., Rubio, A., Sternagel, C., Waldmann, J., Yamada, A.: The termination and complexity competition. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) Proc. 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 11429, pp. 156–166 (2019). https://doi.org/10.1007/978-3-030-17502-3_10
24. Godoy, G., Tiwari, A.: Confluence of shallow right-linear rewrite systems. In: Ong, L. (ed.) Proc. 14th International Conference on Computer Science Logic. Lecture Notes in Computer Science, vol. 3634, pp. 541–556 (2005). https://doi.org/10.1007/11538363_37

25. Godoy, G., Huntingford, E., Tiwari, A.: Termination of rewriting with right-flat rules. In: Baader, F. (ed.) Proc. 18th International Conference on Rewriting Techniques and Applications. Lecture Notes in Computer Science, vol. 4533, pp. 200–213 (2007). https://doi.org/10.1007/978-3-540-73449-9_16
26. Göller, S., Lohrey, M.: The first-order theory of ground tree rewrite graphs. Log. Methods Comput. Sci. (2014). [https://doi.org/10.2168/LMCS-10\(1:7\)2014](https://doi.org/10.2168/LMCS-10(1:7)2014)
27. Gutiérrez, R., Lucas, S., Vitores, M.: Confluence of conditional rewriting in logic form. In: Bojanczyk, M., Chekuri, C. (eds.) Proc. 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science. Leibniz International Proceedings in Informatics, vol. 213, pp. 44:1–44:18 (2021). <https://doi.org/10.4230/LIPIcs.FSTTCS.2021.44>
28. Havlena, V., Holík, L., Lengal, O., Vales, O., Vojnar, T.: Antiprenexing for WSkS: A little goes a long way. In: Albert, E., Kovacs, L. (eds.) Proc. 23rd International Conference on Logic for Programming, Artificial Intelligence, and Reasoning. EPIc Series in Computing, vol. 73, pp. 298–316 (2020). <https://doi.org/10.29007/6bfc>
29. Jiresch, E.: A term rewriting laboratory with systematic and random generation and heuristic test facilities. Master's thesis, Vienna University of Technology (2008)
30. Kaiser, L.: Confluence of right ground term rewriting systems is decidable. In: Sassone, V. (ed.) Proc. 8th International Conference on Foundations of Software Science and Computation Structures. Lecture Notes in Computer Science, vol. 3441, pp. 470–489 (2005). https://doi.org/10.1007/978-3-540-31982-5_30
31. Klarlund, N., Møller, A., Schwartzbach, M.I.: MONA implementation secrets. Int. J. Found. Comput. Sci. **13**(4), 571–586 (2002). <https://doi.org/10.1142/S012905410200128X>
32. Lochmann, A.: Reducing Rewrite Properties to Properties on Ground Terms. Archive of Formal Proofs (2022). https://isa-afp.org/entries/Rewrite_Properties_Reduction.html
33. Lochmann, A., Felgenhauer, B.: First-order theory of rewriting. Archive of Formal Proofs (2022). https://isa-afp.org/entries/FO_Theory_Rewriting.html
34. Lochmann, A., Middeldorp, A.: Formalized proofs of the infinity and normal form predicates in the first-order theory of rewriting. In: Biere, A., Parker, D. (eds.) Proc. 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 12079, pp. 178–194 (2020). https://doi.org/10.1007/978-3-030-45237-7_11
35. Lochmann, A., Felgenhauer, B., Sternagel, C., Thiemann, R., Sternagel, T.: Regular tree relations. Archive of Formal Proofs (2021a). https://www.isa-afp.org/entries/Regular_Tree_Relations.html
36. Lochmann, A., Middeldorp, A., Mitterwallner, F., Felgenhauer, B.: A verified decision procedure for the first-order theory of rewriting for linear variable-separated rewrite systems variable-separated rewrite systems in Isabelle/HOL. In: Hriju, C., Popescu, A. (eds.) Proc. 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, pp. 250–263 (2021b). <https://doi.org/10.1145/3437992.3439918>
37. Lochmann, A., Mitterwallner, F., Middeldorp, A.: Formalized signature extension results for confluence, commutation and unique normal forms. In: Mimram, S., Rocha, C. (eds.) Proc. 10th International Workshop on Confluence, pp. 25–30 (2021)
38. Lochmann, A., Mitterwallner, F., Middeldorp, A.: Formalized signature extension results for equivalence. In: Winkler, S., Rocha, C. (eds.) Proc. 11th International Workshop on Confluence, pp. 42–47 (2022)
39. Marcinkowski, J.: Undecidability of the first order theory of one-step right ground rewriting. In: Comon, H. (ed.) Proc. 8th International Conference on Rewriting Techniques and Applications. Lecture Notes in Computer Science, vol. 1232, pp. 241–253 (1997). https://doi.org/10.1007/3-540-62950-5_75
40. Middeldorp, A.: Approximating dependency graphs using tree automata techniques. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) Proc. 1st International Joint Conference on Automated Reasoning. LNAI, vol. 2083, pp. 593–610 (2001). https://doi.org/10.1007/3-540-45744-5_49
41. Middeldorp, A., Nagele, J., Shintani, K.: Confluence competition 2019. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) Proc. 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 11429, pp. 25–40 (2019). https://doi.org/10.1007/978-3-030-17502-3_2
42. Mitterwallner, F., Lochmann, A., Middeldorp, A., Felgenhauer, B.: Certifying proofs in the first-order theory of rewriting. In: Groote, J.F., Larsen, K.G. (eds.) Proc. 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 12652, pp. 127–144 (2021). https://doi.org/10.1007/978-3-030-72013-1_7
43. Nagaya, T., Toyama, Y.: Decidability for left-linear growing term rewriting systems. Inf. Comput. **178**(2), 499–514 (2002). <https://doi.org/10.1006/inco.2002.3157>
44. Nagele, J., Felgenhauer, B., Middeldorp, A.: CSI: New evidence—a progress report. In: de Moura, L. (ed.) Proc. 26th International Conference on Automated Deduction. LNAI, vol. 10395, pp. 385–397 (2017). https://doi.org/10.1007/978-3-319-63046-5_24

45. Plaisted, D.A.: Polynomial time termination and constraint satisfaction tests. In: Kirchner, C. (ed.) Proc. 5th International Conference on Rewriting Techniques and Applications. Lecture Notes in Computer Science, vol. 690, pp. 405–420 (1993). https://doi.org/10.1007/978-3-662-21551-7_30
46. Rapp, F., Middeldorp, A.: Automating the first-order theory of left-linear right-ground term rewrite systems. In: Kesner, D., Pientka, B. (eds.) Proc. 1st International Conference on Formal Structures for Computation and Deduction. Leibniz International Proceedings in Informatics, vol. 52, pp 36:1–36:12 (2016). <https://doi.org/10.4230/LIPIcs.FSCD.2016.36>
47. Rapp, F., Middeldorp, A.: Confluence properties on open terms in the first-order theory of rewriting. In: Accattoli, B., Tiwari, A. (eds.) Proc. 5th International Workshop on Confluence, pp. 26–30 (2016)
48. Rapp, F., Middeldorp, A.: FORT 2.0. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) Proc. 9th International Joint Conference on Automated Reasoning. LNAI, vol. 10900, pp. 81–88 (2018). https://doi.org/10.1007/978-3-319-94205-6_6
49. Shintani, K., Hirokawa, N.: CoLL: A confluence tool for left-linear term rewrite systems. In: Felty, A.P., Middeldorp, A. (eds.) Proc. 25th International Conference on Automated Deduction. Lecture Notes in Computer Science, vol. 9195, pp. 127–136 (2015). https://doi.org/10.1007/978-3-319-21401-6_8
50. Snyder, W.: A fast algorithm for generating reduced ground rewriting systems from a set of ground equations. *J. Symbol. Comput.* **15**(4), 415–450 (1993). <https://doi.org/10.1006/jsco.1993.1029>
51. Sternagel, C., Sternagel, T.: Certifying confluence of almost orthogonal CTRSs via exact tree automata completion. In: Kesner, D., Pientka, B. (eds.) Proc. 1st International Conference on Formal Structures for Computation and Deduction. Leibniz International Proceedings in Informatics, vol. 52, pp. 29:1–29:16 (2016). <https://doi.org/10.4230/LIPIcs.FSCD.2016.29>
52. Stump, A., Zantema, H., Kimmell, G., Omar, R.E.H.: A rewriting view of simple typing. *Log. Methods Comput. Sci.* (2012). [https://doi.org/10.2168/LMCS-9\(1:4\)2013](https://doi.org/10.2168/LMCS-9(1:4)2013)
53. Thiemann, R., Sternagel, C.: Certification of termination proofs using CeTA. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Proc. 22nd International Conference on Theorem Proving in Higher Order Logics. Lecture Notes in Computer Science, vol. 5674, pp. 452–468 (2009). https://doi.org/10.1007/978-3-642-03359-9_31
54. Treinen, R.: The first-order theory of linear one-step rewriting is undecidable. *Theor. Comput. Sci.* **208**(1–2), 179–190 (1998). [https://doi.org/10.1016/S0304-3975\(98\)00083-8](https://doi.org/10.1016/S0304-3975(98)00083-8)
55. Vorobyov, S.: The undecidability of the first-order theories of one step rewriting in linear canonical systems. *Inf. Comput.* **175**(2), 182–213 (2002). <https://doi.org/10.1006/inco.2002.3151>
56. Zantema, H.: Automatically finding non-confluent examples in term rewriting. In: Hirokawa, N., van Oostrom, V. (eds.) Proc. 2nd International Workshop on Confluence, pp. 11–15 (2013). <http://cl-informatik.uibk.ac.at/iwc/iwc2013.pdf>
57. Zantema, H.: Finding small counterexamples for abstract rewriting properties. *Math. Struct. Comput. Sci.* **28**, 1485–1505 (2018). <https://doi.org/10.1017/S0960129518000221>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.