A Verified Ground Confluence Tool for Linear Variable-Separated Rewrite Systems in Isabelle/HOL

Bertram Felgenhauer
Department of Computer Science
University of Innsbruck
Innsbruck, Austria
bertram.felgenhauer@uibk.ac.at

Aart Middeldorp
Department of Computer Science
University of Innsbruck
Innsbruck, Austria
aart.middeldorp@uibk.ac.at

T. V. H. Prathamesh
Department of Computer Science
University of Innsbruck
Innsbruck, Austria
venkata.turaga@uibk.ac.at

Franziska Rapp
Allgemeines Rechenzentrum Innsbruck
Innsbruck, Austria

Abstract

It is well known that (ground) confluence is a decidable property of ground term rewrite systems, and that this extends to larger classes. Here we present a formally verified ground confluence checker for linear, variable-separated rewrite systems. To this end, we formalize procedures for ground tree transducers and so-called RR_n relations. The ground confluence checker is an important milestone on the way to formalizing the decidability of the first-order theory of ground rewriting for linear, variable-separated rewrite systems. It forms the basis for a formalized confluence checker for left-linear, right-ground systems.

CCS Concepts • **Theory of computation** → **Equational logic and rewriting**; **Tree languages**; **Logic and verification**.

Keywords ground confluence, ground tree transducers, formalization

1 Introduction

First-order term rewriting is a non-deterministic model of computation. The idea is to take a starting term, and repeatedly apply rules of a term rewrite system (TRS for short) until one reaches a normal form where no further reductions are possible. The starting term is the input, and the final term is a result of the computation. Here, a rule is an oriented equation between terms $\ell \to r$, and applying a rule to a term s means to identify a subterm of s that is an instance of ℓ ($s|_p = \ell\sigma$ for a position p of s), and replace it by a corresponding instance of r (resulting in $s[r\sigma]_p$). TRSs arise in the study of equational reasoning (completion), and are also used for program analysis. Confluence is a fundamental property of TRSs, which states that any two terms reachable from a common starting term can also reach a common target term. This property is interesting because it ensures that despite being non-deterministic, normal forms are unique. Confluence is undecidable in general, but is decidable for restricted classes of TRSs. For our work, the difference between *ground confluence*, where rewrite steps are restricted to terms that do not contain variables, and *confluence*, where terms may also contain variables drawn from an infinite set, is important. The two notions coincide if the rules of the TRSs do not contain any variables, i.e., if we are dealing with a *ground* TRS.

Dauchet and Tison [6] were the first to prove decidability of ground confluence for finite, ground TRSs. To this end, they introduced *ground tree transducers* (GTTs for short) consisting of two finite tree automata that, taken together, represent certain binary relations on ground terms, and showed that the parallel rewrite relation $\twoheadrightarrow_{\mathcal{R}}$ associated with a ground rewrite system $\mathcal{R}$ is accepted by a GTT. Since the class of relations accepted by GTTs is effectively closed under inverse, composition, and transitivite closure, ground confluence of $\mathcal{R}$ boils down to an inclusion check between GTT relations ($\twoheadrightarrow_{\mathcal{R}}^{*-1} \cdot \twoheadrightarrow_{\mathcal{R}}^{*} \subseteq \twoheadrightarrow_{\mathcal{R}}^{*} \cdot \twoheadrightarrow_{\mathcal{R}}^{*-1}$). The latter is decided in [6] by an effective transformation from GTT relations

into finite tree automata, using the well-known result that inclusion for regular tree languages is decidable.

The decidability result of [6] was revisited by Dauchet, Tison, Heuillard, and Lescanne in [8], where a different GTT construction is used to represent rewriting in $\mathcal{R}$, resulting in a relation in between $\twoheadrightarrow_{\mathcal{R}}$ and $\twoheadrightarrow_{\mathcal{R}}^*$, and whose transitive closure coincides with $\twoheadrightarrow_{\mathcal{R}}^*$.

Comon, Godoy, and Nieuwenhuis [4] were the first to prove that ground confluence of ground TRSs is decidable in polynomial time. Felgenhauer [9] presented a cubic time algorithm.

In [5] Dauchet *et al.* extended the decision procedure of [6] to left-linear, right-ground TRSs. (The same result was obtained earlier by Oyamaguchi [18] using completely different automata-based techniques.) Compared to [6], in [5] a more direct construction for the transitive closure of GTT relations is given. The latest account is presented in the online textbook on tree automata [3, Chapter 3].

Whereas for ground TRSs, confluence and ground confluence coincide, this is no longer true for non-ground TRSs [21]. Ground confluence is known to be decidable for right-ground [14, 16] and shallow, right-linear [13] TRSs but, due to non-linearity of the left-hand sides of the rewrite rules, the decision procedures use advanced tree automata techniques which go beyond the scope of this paper.

Our interest is the *first-order theory* of rewriting. In this theory properties definable by a first-order formula over ground rewrite predicates like $\rightarrow$ and $\rightarrow^*$ are expressible. This includes ground confluence. In another paper [7], Dauchet and Tison showed that the first-order theory of rewriting is decidable for ground TRSs. The decision procedure (extended to left-linear, right-ground rewrite systems) is implemented in FORT [19, 21]. Besides GTTs and their closure properties, RR_n relations, which are n-ary relations on ground terms that allow an encoding as regular tree languages, play a key role in the decision procedure. Every GTT relation can be represented as an RR_2 relation.

Our long-term aim is to formalize the decision procedure in the proof assistant Isabelle/HOL such that the output of FORT can be certified. (To this end, FORT would emit a sequence of operations on automata that correspond to a formula; the certifier would then compute the corresponding automata using a verified implementation.) In this paper we present a formalization of ground tree transducers and their closure properties. Furthermore, a number of results on the interplay between rewriting and ground tree transducers are formalized, bringing us close to the first formalized proof of the decidability of confluence of ground TRSs.

Our formalization is based on IsaFoR [22].[1] Our own development can be found at [ℭ]/fortissimo/cpp2019/, where [ℭ] abbreviates http://cl-informatik.uibk.ac.at. Furthermore most definitions, theorems, and lemmas directly correspond

to the formalization. These are indicated by the ✔ symbol, which links to a HTML presentation in the PDF version of the paper.

A TRS $\mathcal{R}$ is ground confluent if the inclusion $\uparrow_{\mathcal{R}} \subseteq \downarrow_{\mathcal{R}}$ holds for all ground terms. Here $\uparrow_{\mathcal{R}} = {}_{\mathcal{R}}^* \leftarrow \cdot \rightarrow_{\mathcal{R}}^*$ and $\downarrow_{\mathcal{R}} = \rightarrow_{\mathcal{R}}^* \cdot {}_{\mathcal{R}}^* \leftarrow$. The decision procedures for left-linear, right-ground TRSs mentioned above ([3, 5, 6, 8]) consist of the following ingredients:

1. A GTT $\mathcal{G}$ is constructed from $\mathcal{R}$ which accepts a relation on ground terms in between $\twoheadrightarrow_{\mathcal{R}}$ and $\rightarrow_{\mathcal{R}}^*$.
2. GTT relations are effectively closed under inverse, composition, and transitive closure, allowing $\uparrow_{\mathcal{R}}$ and $\downarrow_{\mathcal{R}}$ to be represented by GTTs $\mathcal{G}_1$ and $\mathcal{G}_2$.
3. The language inclusion $\mathcal{L}(\mathcal{G}_1) \subseteq \mathcal{L}(\mathcal{G}_2)$ is decidable.

The various decision procedures differ in the details.

In the next section we recall basic concepts of term rewriting and tree automata. The three items above are discussed in the subsequent three sections. The discrepancy between confluence and ground confluence is the topic of Section 6. In Section 7 we enhance the formalization to obtain an efficient ground confluence checker for linear, variable-separated TRSs and a confluence checker for left-linear, right-ground TRSs. Experimental results are presented in Section 8 and conclude with mentioning future investigations in Section 9.

2 Preliminaries

We assume familiarity with term rewriting [1] and tree automata [3].

Let $\mathcal{F}$ be a set of function symbols with arities. We assume that $\mathcal{V}$ is an infinite set of variables disjoint from $\mathcal{F}$. The set of terms $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is defined inductively: A term is either a variable $x \in \mathcal{V}$, or $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$ has arity n and $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ for $1 \leqslant i \leqslant n$. We write $Var(t)$ to denote the set of variables in t. A term is ground if it contains no variables; it is linear if each variable occurs at most once in it. The set of ground terms is denoted by $\mathcal{T}(\mathcal{F})$. We write $s \trianglelefteq t$ if s is a subterm of t. A position is a sequence of natural numbers (we use ϵ to denote the empty position) that can be used to address subterms: $s|_\epsilon = s$ and $f(t_1, \ldots, t_n)|_{i.p} = t_i|_p$ if $1 \leqslant i \leqslant n$. We write $t[s]_p$ for the result of replacing the subterm at position p of t by s. A variable position of t is a position p with $t|_p \in \mathcal{V}$; we write $\mathcal{P}os_{\mathcal{V}}(t)$ for the set of variable positions of t. A context C is a term that contains exactly one hole, denoted by the special constant $\square \notin \mathcal{F}$. We write $C[s]$ for the result of replacing the hole by the term s in C. A multi-hole context C may have several occurrences of $\square$, and we write $C[s_1, \ldots, s_n]$ for the result of replacing the n holes of C by $s_1, \ldots, s_n$ from left to right. A substitution σ is a map from variables to terms. Given a term t and substitution σ, $t\sigma$ is defined inductively by $x\sigma = \sigma(x)$ if $x \in \mathcal{V}$ and $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$.

A term rewrite system (TRS) $\mathcal{R}$ is a set of rules $\ell \rightarrow r$ between terms $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$; ℓ is the left-hand side and

---

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

r is the right-hand side of $\ell \to r$. We do not demand that variables in the right-hand side do also occur in the left-hand side, and allow left-hand sides to be variables. A TRS is left-linear (left-ground) if its left-hand sides are linear (ground); it is right-linear (right-ground) if the right-hand sides are linear (ground); it is linear (ground) if it is both left- and right-linear (-ground). A variable-separate TRS $\mathcal{R}$ is one such that the variables of ℓ and r are disjoint for all $\ell \to r \in \mathcal{R}$. A TRS $\mathcal{R}$ gives rise to a rewrite relation $\to_{\mathcal{R}}$, where $C[\ell\sigma] \to_{\mathcal{R}} C[r\sigma]$ for any context C, rule $\ell \to r$ and substitution σ. Given a (rewrite) relation $\to$, we write $\leftarrow, \leftrightarrow, \to^=, \to^+, \to^*, \mathrel{\mathpalette\@twoheadrightarrow\relax}$ for the inverse, the symmetric closure, the reflexive closure, the transitive closure, the reflexive transitive closure, and the parallel closure of $\to$. The latter is defined inductively by

$$\frac{x \in \mathcal{V}}{x \mathrel{+\!\!\!\to} x} \qquad \frac{s \to t}{s \mathrel{+\!\!\!\to} t} \qquad \frac{(s_i \mathrel{+\!\!\!\to} t_i)_{i=1}^n}{f(s_1, \ldots, s_n) \mathrel{+\!\!\!\to} f(t_1, \ldots, t_n)}$$

Alternatively, we write R^- for the inverse of a relation (or TRS) R. The composition operator for (rewrite) relations is $\cdot$. We define convertibility $\leftrightarrow^*$, joinability $\downarrow = \to^* \cdot {}^*\!\leftarrow$, and meetability $\uparrow = {}^*\!\leftarrow \cdot \to^*$. A relation $\to$ is confluent if $\uparrow \subseteq \downarrow$ (or, equivalently, $\leftrightarrow^* \subseteq \downarrow$).

Remark 2.1. There are recurring complications in the formalization of the notion of rewriting. IsaFoR provides notions of rewrite steps $\to_{\mathcal{R}}$, parallel steps $\mathrel{+\!\!\!\to}_{\mathcal{R}}$ and a few others. However, these do not restrict the signature (except via the type of variables and function symbols), so rewriting works on terms with variables and, typically, many extra function symbols. Consequently, we have to explicitly restrict rewriting steps to ground terms and to a particular signature inside the formalization. In this paper, we follow the style from the literature which leaves these details largely implicit.

A tree automaton is a triple $\mathcal{A} = (Q, Q_f, \Delta)$, where Q is a set of states disjoint from $\mathcal{F}$, Q_f is a set of final states ($Q_f \subseteq Q$), and Δ is a set of transitions, which come in two shapes: ordinary transitions $f(q_1, \ldots, q_n) \to q$ where $f \in \mathcal{F}$ is of arity n and $q, q_1, \ldots, q_n \in Q$, and ϵ-transitions $p \to q$ with $p, q \in Q$. The transitions form a ground TRS, and we write $\to_{\mathcal{A}}$ for $\to_\Delta$. The language accepted by $\mathcal{A}$ in state q is $\mathcal{L}(\mathcal{A}, q) = \{t \mid t \in \mathcal{T}(\mathcal{F}) \text{ and } t \to_{\mathcal{A}}^* q\}$, and the language accepted by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \bigcup_{q \in Q_f} \mathcal{L}(\mathcal{A}, q)$. A ground tree transducer $\mathcal{G}$ is a pair of tree automata $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ over the same set of states Q. It defines a relation on ground terms,

$$\mathcal{L}(\mathcal{G}) = \{(s, t) \mid s, t \in \mathcal{T}(\mathcal{F}) \text{ and } s \xrightarrow{*}_{\mathcal{A}} \cdot \xleftarrow{*}_{\mathcal{B}} t\}$$

For tree automata $\mathcal{A}$ and $\mathcal{A}'$ we write $\mathcal{A} \subseteq \mathcal{A}'$ if each transition of $\mathcal{A}$ is also a transition of $\mathcal{A}'$, and each final state of $\mathcal{A}$ is a final state of $\mathcal{A}'$. For GTTs we define $(\mathcal{A}, \mathcal{B}) \subseteq (\mathcal{A}', \mathcal{B}')$ as $\mathcal{A} \subseteq \mathcal{A}'$ and $\mathcal{B} \subseteq \mathcal{B}'$, but ignoring the final states of the tree automata.

Proposition 2.1. *Let $\mathcal{A}$ and $\mathcal{A}'$ be tree automata and $\mathcal{G}$ and $\mathcal{G}'$ be GTTs. If $\mathcal{A} \subseteq \mathcal{A}'$ then $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$ (part of IsaFoR) and if $\mathcal{G} \subseteq \mathcal{G}'$ then $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{G}')$.* ✔

3 Modeling Rewriting by GTTs

Throughout this paper we deal with linear, variable-separated TRSs. We use the acronym LV-TRS to denote such TRSs. As mentioned in the introduction, there are several ways to associate a GTT $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ with an LV-TRS $\mathcal{R}$. The one in [6] uses for each rewrite rule $\ell \to r$ 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 $\mathrel{+\!\!\!\to}_{\mathcal{R}}$. The second way to associate a GTT with an LV-TRS $\mathcal{R}$ originates from Dauchet *et al.* [5]. The resulting GTT accepts a relation in between $\mathrel{+\!\!\!\to}_{\mathcal{R}}$ and $\to_{\mathcal{R}}^*$. In [11, Theorem 1] we formalized this result for ground TRSs. Here we present a variant that accepts $\mathrel{+\!\!\!\to}_{\mathcal{R}}$ precisely, which is achieved by keeping the number of shared states to a minimum.

Definition 3.1. Let $\mathcal{R}$ be an LV-TRS. We denote the set of left-hand (right-hand) sides of the rewrite rules in $\mathcal{R}$ by $\mathsf{lhs}(\mathcal{R})$ ($\mathsf{rhs}(\mathcal{R})$). Given a set of terms T, we write $s \trianglelefteq T$ if s is a subterm of some term in T. Given s we write $\hat{s}$ for the ground term obtained from s by replacing each variable by a designated (fresh) constant $*$.[2] The GTT will have states $\langle \hat{s} \rangle$ for each $s \trianglelefteq \mathsf{lhs}(\mathcal{R})$ and $[\hat{s}]$ for each $s \trianglelefteq \mathsf{rhs}(\mathcal{R})$. The set Δ_{lhs} consists of the transitions

$$f(\langle \widehat{t_1} \rangle, \ldots, \langle \widehat{t_n} \rangle) \to \langle \widehat{f(t_1, \ldots, t_n)} \rangle$$

for every $f(t_1, \ldots, t_n) \trianglelefteq \mathsf{lhs}(\mathcal{R})$ and

$$f(\langle * \rangle, \ldots, \langle * \rangle) \to \langle * \rangle$$

for every function symbol f in the signature of $\mathcal{R}$. The set Δ_{rhs} is defined similarly, using $\mathsf{rhs}(\mathcal{R})$ instead of $\mathsf{lhs}(\mathcal{R})$ for generating the rules and square brackets $[\cdot]$ rather than angular brackets $\langle \cdot \rangle$ in states to ensure that Δ_{lhs} and Δ_{rhs} do not share states. We define the GTT $\mathcal{G}(\mathcal{R}) = (\mathcal{A}, \mathcal{B})$ with ✔

$$\mathcal{A} = \Delta_{\mathsf{lhs}} \cup \{\langle \hat{\ell} \rangle \to [\hat{r}] \mid \ell \to r \in \mathcal{R}\} \qquad ✔$$
$$\mathcal{B} = \Delta_{\mathsf{rhs}} \qquad ✔$$

Theorem 3.2. $\mathcal{L}(\mathcal{G}(\mathcal{R})) = \mathrel{+\!\!\!\to}_{\mathcal{R}}$ ✔

Example 3.3. We illustrate the construction on the LV-TRS $\mathcal{R}$ consisting of the rules $\mathsf{a} \to \mathsf{f}(\mathsf{a})$, $\mathsf{a} \to \mathsf{b}$, and $\mathsf{f}(x) \to \mathsf{g}(\mathsf{a}, \mathsf{b})$. We construct the GTT $\mathcal{G}(\mathcal{R}) = (\mathcal{A}, \mathcal{B})$ with $\mathcal{A}$ consisting of the rules

$$\begin{array}{lll}
\mathsf{a} \to \langle * \rangle & \mathsf{f}(\langle * \rangle) \to \langle * \rangle & \mathsf{g}(\langle * \rangle, \langle * \rangle) \to \langle * \rangle \\
\mathsf{b} \to \langle * \rangle & \mathsf{a} \to \langle \mathsf{a} \rangle & \mathsf{f}(\langle * \rangle) \to \langle \mathsf{f}(*) \rangle \\
\langle \mathsf{a} \rangle \to [\mathsf{f}(\mathsf{a})] & \langle \mathsf{a} \rangle \to [\mathsf{b}] & \langle \mathsf{f}(*) \rangle \to [\mathsf{g}(\mathsf{a}, \mathsf{b})]
\end{array}$$

and $\mathcal{B}$ consisting of the rules

$$\begin{array}{ll}
\mathsf{a} \to [\mathsf{a}] & \mathsf{f}([\mathsf{a}]) \to [\mathsf{f}(\mathsf{a})] \\
\mathsf{b} \to [\mathsf{b}] & \mathsf{g}([\mathsf{a}], [\mathsf{b}]) \to [\mathsf{g}(\mathsf{a}, \mathsf{b})]
\end{array}$$

---

[2]In the formalization, we use an option type for adding $*$ to the existing signature, avoiding the problem of creating fresh names.

$$\frac{p \to_{\mathcal{A}} q \quad p \rightsquigarrow r}{q \rightsquigarrow r} \qquad \frac{p \rightsquigarrow q \quad q \to_{\mathcal{B}} r}{p \rightsquigarrow r}$$

$$\frac{f(p_1, \ldots, p_n) \to_{\mathcal{A}} p \quad f(q_1, \ldots, q_n) \to_{\mathcal{B}} q \quad (p_1 \rightsquigarrow q_i)_{i=1}^n}{p \rightsquigarrow q}$$

Figure 1. ϵ-transitions for GTT composition

Note that we have omitted the following transitions from $\mathcal{B}$, because they do not contribute to the GTT relation:

$$\mathsf{a} \to [*] \quad \mathsf{b} \to [*] \quad \mathsf{f}([*]) \to [*] \quad \mathsf{g}([*],[*]) \to [*]$$

(This is an instance of *trimming*, cf. Remark 4.1.) The GTT $\mathcal{G}(\mathcal{R})$ accepts $\twoheadrightarrow_{\mathcal{R}}$; for example, the parallel step $\mathsf{g}(\mathsf{f}(\mathsf{b}), \mathsf{a}) \twoheadrightarrow_{\mathcal{R}}$ $\mathsf{g}(\mathsf{g}(\mathsf{a}, \mathsf{b}), \mathsf{b})$ is accepted by $\mathsf{g}(\mathsf{f}(\mathsf{b}), \mathsf{a}) \to_{\mathcal{A}}^* \mathsf{g}(\langle \mathsf{f}(*)\rangle, \langle \mathsf{a}\rangle) \to_{\mathcal{A}}^*$ $\mathsf{g}([\mathsf{g}(\mathsf{a}, \mathsf{b})], [\mathsf{b}]) {}_{\mathcal{B}}^* \!\!\leftarrow \mathsf{g}(\mathsf{g}(\mathsf{a}, \mathsf{b}), \mathsf{b})$.

4 Formalizing GTT Closure Properties

The primary reason why GTT relations are of interest is because they are closed under inverse, composition and transitive closure. It is noteworthy that despite the fact that GTT languages depend on the signature $\mathcal{F}$, the constructions underlying these closure properties only depend on the GTTs themselves. So for this section, we assume a fixed signature $\mathcal{F}$ such that every function symbol that occurs in the considered GTTs is an element $\mathcal{F}$. We have formalized these three closure properties. The inverse is obtained by swapping the two automata that constitute the GTT. ✔

For composing GTT relations, the following construction is used.

Definition 4.1. Let $\mathcal{G}_1 = (\mathcal{A}_1, \mathcal{B}_1)$ and $\mathcal{G}_2 = (\mathcal{A}_2, \mathcal{B}_2)$. We let $\hat{\circ}(\mathcal{G}_1, \mathcal{G}_2) =$

$$(\mathcal{A}_1 \cup \mathcal{A}_2 \cup \Delta_\epsilon(\mathcal{B}_1, \mathcal{A}_2), \mathcal{B}_1 \cup \mathcal{B}_2 \cup \Delta_\epsilon(\mathcal{A}_2, \mathcal{B}_1)) \quad ✔$$

where $\Delta_\epsilon(\mathcal{A}, \mathcal{B})$ is the set of ϵ-transitions $\rightsquigarrow$ defined inductively by the rules in Figure 1. ✔

This is closely related to the construction as used by Dauchet *et al.* [7], which defines Δ_ϵ in a different, but equivalent way, as shown by the next lemma.

Lemma 4.2. $\Delta_\epsilon(\mathcal{A}, \mathcal{B}) = \{(p, q) \mid p {}_{\mathcal{A}}^* \!\!\leftarrow t \to_{\mathcal{B}}^* q \text{ for some term } t \in \mathcal{T}(\mathcal{F})\}$ ✔

Intuitively, the first two rules of Figure 1 deal with ϵ-transitions from $\mathcal{A}$ and $\mathcal{B}$, whereas the last rule correspond to a decomposition of t in $p {}_{\mathcal{A}}^* \!\!\leftarrow t \to_{\mathcal{B}}^* q$ into root symbol and arguments.

Theorem 4.3. *If $\mathcal{G}_1$ and $\mathcal{G}_2$ are GTTs with disjoint states then $\mathcal{L}(\hat{\circ}(\mathcal{G}_1, \mathcal{G}_2)) = \mathcal{L}(\mathcal{G}_1) \circ \mathcal{L}(\mathcal{G}_2)$.* ✔

Proof. In the formalization, we follow the lines of the proof in [3, Proposition 3.2.16]. We show that

1. $\mathcal{L}(\mathcal{G}_1) \circ \mathcal{L}(\mathcal{G}_2) \subseteq \mathcal{L}(\hat{\circ}(\mathcal{G}_1, \mathcal{G}_2))$ ✔

$$\frac{p \to_{\mathcal{A}} q \quad p \rightsquigarrow r}{q \rightsquigarrow r} \qquad \frac{p \rightsquigarrow q \quad q \to_{\mathcal{B}} r}{p \rightsquigarrow r} \qquad \frac{p \rightsquigarrow q \quad q \rightsquigarrow r}{p \rightsquigarrow r}$$

$$\frac{f(p_1, \ldots, p_n) \to_{\mathcal{A}} p \quad f(q_1, \ldots, q_n) \to_{\mathcal{B}} q \quad (p_1 \rightsquigarrow q_i)_{i=1}^n}{p \rightsquigarrow q}$$

Figure 2. ϵ-transitions for GTT transitive closure

2. $\mathcal{L}(\hat{\circ}(\mathcal{G}_1, \mathcal{G}_2)) \subseteq \mathcal{L}(\mathcal{G}_1) \circ \mathcal{L}(\mathcal{G}_2)$ ✔

The assumption that $\mathcal{G}_1$ and $\mathcal{G}_2$ do not share states is only used in the second statement. □

Definition 4.4. Let $\mathcal{G} = (\mathcal{A}, \mathcal{B})$. We let

$$\hat{+}(\mathcal{G}) = (\mathcal{A} \cup \Delta_+(\mathcal{B}, \mathcal{A}), \mathcal{B} \cup \Delta_+(\mathcal{A}, \mathcal{B})) \quad ✔$$

where $\Delta_+(\mathcal{A}, \mathcal{B}) = \rightsquigarrow$ is defined inductively by the rules in Figure 2. ✔

This definition differs from the literature, which is based on—in essence—iterating the GTT composition construction until a fixed point is reached. The inductive definition here benefits both the correctness proofs and the implementation (see Section 7).

To see the benefit for the correctness proof, note that the inference rules for the ϵ-transitions for the transitive closure (Figure 2) differ from those for the GTT composition (Figure 1) only in the addition of a transitivity rule for $\rightsquigarrow$. This observation allows us to reuse results from the GTT composition formalization for the transitive closure in the following lemma.

Lemma 4.5. $\mathcal{L}(\mathcal{G})^+ \subseteq \mathcal{L}(\hat{+}(\mathcal{G}))$ ✔

Proof. It suffices to prove that $\mathcal{L}(\hat{+}(\mathcal{G}))$ extends $\mathcal{L}(\mathcal{G})$, and that $\mathcal{L}(\hat{+}(\mathcal{G}))$ is transitive.

- For the extension part, note that $\mathcal{G} \subseteq \hat{+}(\mathcal{G})$, and hence $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\hat{+}(\mathcal{G}))$ as claimed. ✔
- For transitivity, note that $\hat{\circ}(\hat{+}(\mathcal{G}), \hat{+}(\mathcal{G})) \subseteq \hat{+}(\mathcal{G})$ (which Isabelle proves automatically) so that

$$\mathcal{L}(\hat{+}(\mathcal{G})) \circ \mathcal{L}(\hat{+}(\mathcal{G})) = \mathcal{L}(\hat{\circ}(\hat{+}(\mathcal{G}), \hat{+}(\mathcal{G}))) \subseteq \mathcal{L}(\hat{+}(\mathcal{G})) \quad ✔$$

using Theorem 4.3 in the first step. □

For the converse inclusion, $\mathcal{L}(\hat{+}(\mathcal{G})) \subseteq \mathcal{L}(\mathcal{G})^+$, there is no such shortcut. The following lemma is crucial in establishing this result.

Lemma 4.6. *If $(p, q) \in \Delta_+(\mathcal{A}, \mathcal{B})$ then there are terms $s \in \mathcal{L}(\mathcal{A}, p)$, $t \in \mathcal{L}(\mathcal{A}, q)$ with $(s, t) \in \mathcal{L}((\mathcal{B}, \mathcal{A}))^+$.* ✔

Proof. We write $s \; [\mathcal{L}((\mathcal{B}, \mathcal{A}))]^+ \; t$ for $(s, t) \in \mathcal{L}((\mathcal{B}, \mathcal{A}))^+$. The proof is by induction on $\rightsquigarrow$ in Figure 2. The most interesting case is the transitivity rule. If $p \rightsquigarrow r$ is derived from $p \rightsquigarrow q$ and $q \rightsquigarrow r$ then by the induction hypothesis,

$$p \xleftarrow{*}_{\mathcal{A}} s \; [\mathcal{L}((\mathcal{B}, \mathcal{A}))]^+ \; t \xrightarrow{*}_{\mathcal{B}} q \xleftarrow{*}_{\mathcal{A}} u \; [\mathcal{L}((\mathcal{B}, \mathcal{A}))]^+ \; v \xrightarrow{*}_{\mathcal{B}} r$$

which implies $t \; [\mathcal{L}((\mathcal{B}, \mathcal{A}))] \; u$, and consequently

$$p \overset{*}{\underset{\mathcal{A}}{\leftarrow}} s \; [\mathcal{L}((\mathcal{B}, \mathcal{A}))]^+ \; t \overset{*}{\underset{\mathcal{B}}{\rightarrow}} r$$

as desired. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

Lemma 4.7. *If $\hat{+}(\mathcal{A}, \mathcal{B}) = (\mathcal{A}_+, \mathcal{B}_+)$ and $s \in \mathcal{L}(\mathcal{A}_+, p)$ then $(s, t) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))^+$ for some term $t \in \mathcal{L}(\mathcal{A}, p)$.* ✔

Proof. We proceed by induction on the length of the reduction $s \rightarrow^*_{\mathcal{A}_+} p$. If the last step is an ϵ-transition $q \rightarrow p$ then the induction hypothesis yields a ground term u with $(s, u) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))^+$ and $u \in \mathcal{L}(\mathcal{A}, q)$. If $q \rightarrow p$ is a transition from $\mathcal{A}$ then $u \in \mathcal{L}(\mathcal{A}, p)$, and we conclude by letting $t = u$; otherwise, $q \rightarrow p$ must come from $\Delta_+(\mathcal{B}, \mathcal{A})$, and using Lemma 4.6 we obtain ground terms v and w with $v \in \mathcal{L}(\mathcal{B}, q)$, $w \in \mathcal{L}(\mathcal{A}, p)$, and $(v, w) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))^+$. This implies $(u, v) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))$ and thus $(s, w) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))^+$ by transitivity. Letting $t = w$ gives the desired result. If the last step is not an ϵ-transition, then it must be a transition $f(p_1, \ldots, p_n) \rightarrow p$ from $\mathcal{A}$, and we have $s = f(s_1, \ldots, s_n)$ for suitable $s_1, \ldots, s_n$. We apply the induction hypothesis to each argument position, resulting in $t_1, \ldots, t_n$ with $(s_i, t_i) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))^+$ and $t_i \in \mathcal{L}(\mathcal{A}, p_i)$ for $1 \leqslant i \leqslant n$. Let $t = f(t_1, \ldots, t_n)$. We have $t \in \mathcal{L}(\mathcal{A}, p)$. Since $\mathcal{L}((\mathcal{A}, \mathcal{B}))^+$ is transitive and closed under contexts, we obtain $(s, t) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))^*$. Since $\mathcal{L}((\mathcal{A}, \mathcal{B}))$ is reflexive, we actually have $(s, t) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))^+$ as desired. $\qquad\square$

Lemma 4.8. $\mathcal{L}(\hat{+}(\mathcal{G})) \subseteq \mathcal{L}(\mathcal{G})^+$ ✔

Proof. Let $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ and $\hat{+}(\mathcal{G}) = (\mathcal{A}_+, \mathcal{B}_+)$. Assume that $(s, t) \in \mathcal{L}(\hat{+}(\mathcal{G}))$. Hence there are a multihole context C, terms $s_1, \ldots, s_n, t_1, \ldots, t_n$, and states $q_1, \ldots, q_n$ such that $s = C[s_1, \ldots, s_n]$, $t = C[t_1, \ldots, t_n]$, and $s_i \in \mathcal{L}(\mathcal{A}_+, q_i)$ and $t_i \in \mathcal{L}(\mathcal{B}_+, q_i)$ for $1 \leqslant i \leqslant n$. Applying Lemma 4.7 to $\hat{+}(\mathcal{A}, \mathcal{B})$ and $\hat{+}(\mathcal{B}, \mathcal{A})$ yields terms $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ with $(s_i, u_i) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))^+$, $u_i \in \mathcal{L}(\mathcal{A}, q_i)$, $(t_i, v_i) \in \mathcal{L}((\mathcal{B}, \mathcal{A}))^+$, $v_i \in \mathcal{L}(\mathcal{B}, q_i)$ for $1 \leqslant i \leqslant n$. This implies $(u_i, v_i) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))$ and $(v_i, t_i) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))^+$, so that $(s_i, t_i) \in \mathcal{L}((\mathcal{A}, \mathcal{B}))^+ = \mathcal{L}(\mathcal{G})^+$. Transitivity, reflexivity, and closure under context of $\mathcal{L}(\mathcal{G})^+$ yield $(s, t) \in \mathcal{L}(\mathcal{G})^+$, as claimed. $\qquad\square$

As a corollary of Lemmata 4.5 and 4.8, we obtain correctness of the transitive closure construction.

Theorem 4.9. $\mathcal{L}(\hat{+}(\mathcal{G})) = \mathcal{L}(\mathcal{G})^+$ ✔

Remark 4.1. We can also simplify GTTs $(\mathcal{A}, \mathcal{B})$, restricting each of the two constituent automata to states that are reachable (p is reachable in $\mathcal{A}$ if $\mathcal{L}(\mathcal{A}, p)$ is non-empty) and productive (p is productive in $\mathcal{A}$ if there are a ground context C and a common state q of $\mathcal{A}$ and $\mathcal{B}$ with $C[p] \rightarrow^*_{\mathcal{A}} q$). We have proved that this procedure does not affect the GTT relation. ✔

Example 4.10. Continuing Example 3.3, we demonstrate the transitive closure computation. To this end, we compute

$\Delta_+(\mathcal{A}, \mathcal{B})$ using the rules from Figure 2. We list the inferences to the left, with justifications to the right:

$\langle * \rangle \rightsquigarrow [\mathrm{a}]$	$\langle * \rangle \leftarrow \mathrm{a} \rightarrow [\mathrm{a}]$
$\langle * \rangle \rightsquigarrow [\mathrm{b}]$	$\langle * \rangle \leftarrow \mathrm{b} \rightarrow [\mathrm{b}]$
$\langle * \rangle \rightsquigarrow [\mathrm{f}(\mathrm{a})]$	$\langle * \rangle \leftarrow \mathrm{f}(\langle * \rangle) \rightsquigarrow \mathrm{f}([\mathrm{a}]) \rightarrow [\mathrm{f}(\mathrm{a})]$
$\langle * \rangle \rightsquigarrow [\mathrm{g}(\mathrm{a}, \mathrm{b})]$	$\langle * \rangle \leftarrow \mathrm{g}(\langle * \rangle, \langle * \rangle) \rightsquigarrow^* \mathrm{g}([\mathrm{a}], [\mathrm{b}])$
	$\rightarrow [\mathrm{g}(\mathrm{a}, \mathrm{b})]$
$\langle \mathrm{a} \rangle \rightsquigarrow [\mathrm{a}]$	$\langle \mathrm{a} \rangle \leftarrow \mathrm{a} \rightarrow [\mathrm{a}]$
$\langle \mathrm{f}(*) \rangle \rightsquigarrow [\mathrm{f}(\mathrm{a})]$	$\langle * \rangle \leftarrow \mathrm{f}(\langle * \rangle) \rightsquigarrow \mathrm{f}([\mathrm{a}]) \rightarrow [\mathrm{f}(\mathrm{a})]$
$[\mathrm{b}] \rightsquigarrow [\mathrm{a}]$	$[\mathrm{b}] \leftarrow \langle \mathrm{a} \rangle \rightsquigarrow [\mathrm{a}]$
$[\mathrm{f}(\mathrm{a})] \rightsquigarrow [\mathrm{a}]$	$[\mathrm{f}(\mathrm{a})] \leftarrow \langle \mathrm{a} \rangle \rightsquigarrow [\mathrm{a}]$
$\langle \mathrm{f}(*) \rangle \rightsquigarrow [\mathrm{a}]$	$\langle \mathrm{f}(*) \rangle \rightsquigarrow [\mathrm{f}(\mathrm{a})] \rightsquigarrow [\mathrm{a}]$
$[\mathrm{g}(\mathrm{a}, \mathrm{b})] \rightsquigarrow [\mathrm{a}]$	$[\mathrm{g}(\mathrm{a}, \mathrm{b})] \leftarrow \langle \mathrm{f}(*) \rangle \rightsquigarrow [\mathrm{a}]$
$[\mathrm{g}(\mathrm{a}, \mathrm{b})] \rightsquigarrow [\mathrm{f}(\mathrm{a})]$	$[\mathrm{g}(\mathrm{a}, \mathrm{b})] \leftarrow \langle \mathrm{f}(*) \rangle \rightsquigarrow [\mathrm{f}(\mathrm{a})]$

Further inferences like $\langle * \rangle \rightsquigarrow [\mathrm{b}] \rightsquigarrow [\mathrm{a}]$ do not produce new elements of $\Delta_+(\mathcal{A}, \mathcal{B})$. We do not have to compute $\Delta_+(\mathcal{B}, \mathcal{A})$ because $\Delta_+(\mathcal{B}, \mathcal{A}) = \Delta_+(\mathcal{A}, \mathcal{B})^-$.

After pruning superfluous transitions, we obtain the GTT $\mathcal{G}_+(\mathcal{R}) = (\mathcal{A}_+, \mathcal{B}_+)$ accepting $\rightarrow^*_{\mathcal{R}}$, where $\mathcal{A}_+$ consists of $\mathcal{A}$ and the additional ϵ-transitions

$$[\mathrm{b}] \rightarrow \langle * \rangle \qquad [\mathrm{f}(\mathrm{a})] \rightarrow [\mathrm{g}(\mathrm{a}, \mathrm{b})] \qquad [\mathrm{g}(\mathrm{a}, \mathrm{b})] \rightarrow [*]$$
$$[\mathrm{f}(\mathrm{a})] \rightarrow [*] \qquad [\mathrm{f}(\mathrm{a})] \rightarrow [\mathrm{f}(*)]$$

and $\mathcal{B}_+$ consists of $\mathcal{B}$ and the ϵ-transitions

$$[\mathrm{b}] \rightarrow [\mathrm{a}] \qquad [\mathrm{g}(\mathrm{a}, \mathrm{b})] \rightarrow [\mathrm{a}] \qquad [\mathrm{g}(\mathrm{a}, \mathrm{b})] \rightarrow [\mathrm{f}(\mathrm{a})]$$
$$[\mathrm{f}(\mathrm{a})] \rightarrow [\mathrm{a}]$$

We have $\mathrm{a} \rightarrow^*_{\mathcal{R}} \mathrm{g}(\mathrm{f}(\mathrm{b}), \mathrm{b})$ which is witnessed by $\mathcal{G}_+(\mathcal{R})$ as $\mathrm{a} \rightarrow^*_{\mathcal{A}_+} [\mathrm{f}(\mathrm{a})] \rightarrow_{\mathcal{A}_+} [\mathrm{g}(\mathrm{a}, \mathrm{b})] \; {}_{\mathcal{B}_+}{\overset{*}{\leftarrow}} \; \mathrm{g}(\mathrm{f}(\mathrm{b}), \mathrm{b})$.

Example 4.11. We demonstrate the composition construction, continuing Example 4.10. In order to compute $\downarrow_{\mathcal{R}}$ as a GTT relation, we compose the two GTTs $(\mathcal{A}_+, \mathcal{B}_+)$ and $(\mathcal{B}'_+, \mathcal{A}'_+)$, where $(\mathcal{B}'_+, \mathcal{A}'_+)$ is a renamed version of $(\mathcal{B}_+, \mathcal{A}_+)$ obtained by underlining every state, accepting $\rightarrow^*_{\mathcal{R}}$ and ${}_{\mathcal{R}}{\overset{*}{\leftarrow}}$. We compute $\Delta_\epsilon(\mathcal{B}_+, \mathcal{B}'_+)$ using the inference rules from Figure 1. We list inferences to the left with their justifications to the right:

$[\mathrm{a}] \rightsquigarrow \underline{[\mathrm{a}]}$	$[\mathrm{a}] \leftarrow \mathrm{a} \rightarrow \underline{[\mathrm{a}]}$
$[\mathrm{b}] \rightsquigarrow \underline{[\mathrm{b}]}$	$[\mathrm{b}] \leftarrow \mathrm{b} \rightarrow \underline{[\mathrm{b}]}$
$[\mathrm{a}] \rightsquigarrow \underline{[\mathrm{b}]}$	$[\mathrm{a}] \leftarrow [\mathrm{b}] \rightsquigarrow \underline{[\mathrm{b}]}$
$[\mathrm{b}] \rightsquigarrow \underline{[\mathrm{a}]}$	$[\mathrm{b}] \rightsquigarrow \underline{[\mathrm{b}]} \rightarrow \underline{[\mathrm{a}]}$
$[\mathrm{f}(\mathrm{a})] \rightsquigarrow \underline{[\mathrm{f}(\mathrm{a})]}$	$[\mathrm{f}(\mathrm{a})] \leftarrow \mathrm{f}([\mathrm{a}]) \rightsquigarrow \mathrm{f}(\underline{[\mathrm{a}]}) \rightarrow \underline{[\mathrm{f}(\mathrm{a})]}$
$[\mathrm{a}] \rightsquigarrow \underline{[\mathrm{f}(\mathrm{a})]}$	$[\mathrm{a}] \leftarrow [\mathrm{f}(\mathrm{a})] \rightsquigarrow \underline{[\mathrm{f}(\mathrm{a})]}$
$[\mathrm{f}(\mathrm{a})] \rightsquigarrow \underline{[\mathrm{a}]}$	$[\mathrm{f}(\mathrm{a})] \rightsquigarrow \underline{[\mathrm{f}(\mathrm{a})]} \rightarrow \underline{[\mathrm{a}]}$
$[\mathrm{g}(\mathrm{a}, \mathrm{b})] \rightsquigarrow \underline{[\mathrm{g}(\mathrm{a}, \mathrm{b})]}$	$[\mathrm{g}(\mathrm{a}, \mathrm{b})] \leftarrow \mathrm{g}([\mathrm{a}], [\mathrm{b}]) \rightsquigarrow^* \mathrm{g}(\underline{[\mathrm{a}]}, \underline{[\mathrm{b}]})$
	$\rightarrow \underline{[\mathrm{g}(\mathrm{a}, \mathrm{b})]}$

$$[a] \rightsquigarrow \underline{[g(a,b)]} \qquad [a] \leftarrow [g(a,b)] \rightsquigarrow \underline{[g(a,b)]}$$
$$[f(a)] \rightsquigarrow \underline{[g(a,b)]} \qquad [f(a)] \leftarrow [g(a,b)] \rightsquigarrow \underline{[g(a,b)]}$$
$$[g(a,b)] \rightsquigarrow \underline{[a]} \qquad [g(a,b)] \rightsquigarrow \underline{[g(a,b)]} \rightarrow \underline{[a]}$$
$$[g(a,b)] \rightsquigarrow \underline{[f(a)]} \qquad [g(a,b)] \rightsquigarrow \underline{[g(a,b)]} \rightarrow \underline{[f(a)]}$$

After trimming, we obtain as the result $\mathcal{G}_\downarrow(\mathcal{R}) = (\mathcal{A}_\downarrow, \mathcal{B}_\downarrow)$, where $\mathcal{A}_\downarrow$ consists of $\mathcal{A}_+, \mathcal{B}_+$ and the ϵ-transitions

$$[b] \rightarrow \underline{[a]} \quad [g(a,b)] \rightarrow \underline{[f(a)]} \qquad [f(a)] \rightarrow \underline{[f(a)]}$$
$$[b] \rightarrow \underline{[b]} \qquad [f(a)] \rightarrow \underline{[g(a,b)]} \quad [g(a,b)] \rightarrow \underline{[a]}$$
$$[f(a)] \rightarrow \underline{[a]} \quad [g(a,b)] \rightarrow \underline{[g(a,b)]}$$

The automaton $\mathcal{B}_\downarrow$ is similar to $\mathcal{A}_\downarrow$ but with every underlined state replaced by the corresponding non-underlined state and vice-versa. This is not surprising because the relation $\downarrow_\mathcal{R}$ is symmetric. We have $(g(b,b), b) \notin \mathcal{L}(\mathcal{G}_\downarrow(\mathcal{R}))$ as $g(b,b) \rightarrow^*_{\mathcal{A}_\downarrow} \{g(p,q) \mid p, q \in \{\langle * \rangle, \underline{[a]}, \underline{[b]}\}\} \cup \{\langle * \rangle, \underline{[g(a,b)]}, \underline{[a]}, \underline{[f(a)]}\}$ and $b \rightarrow^*_{\mathcal{B}_\downarrow} \{\underline{\langle * \rangle}, [a], [b]\}$ do not have a common reduct.

The computation of a GTT that accepts $\uparrow_\mathcal{R}$ is more tedious. The set $\Delta_\epsilon(\mathcal{A}'_+, \mathcal{A}_+)$ has 36 elements. After trimming, we obtain as the result the GTT $(\mathcal{A}_\downarrow, \mathcal{B}_\downarrow)$, where $\mathcal{A}_\downarrow$ consists of $\mathcal{A}_+, \mathcal{B}'_+$ and the additional ϵ-transitions

$$\underline{[b]} \rightarrow \langle * \rangle \qquad \underline{[b]} \rightarrow \langle f(*) \rangle \qquad \underline{[g(a,b)]} \rightarrow [b]$$
$$\underline{[b]} \rightarrow \langle a \rangle \qquad \underline{[b]} \rightarrow [f(a)] \qquad \underline{[g(a,b)]} \rightarrow \langle f(*) \rangle$$
$$\underline{[b]} \rightarrow [b] \qquad \underline{[b]} \rightarrow [g(a,b)] \qquad \underline{[g(a,b)]} \rightarrow [f(a)]$$
$$\underline{[f(a)]} \rightarrow \langle * \rangle \quad \underline{[f(a)]} \rightarrow [f(a)] \qquad \underline{[g(a,b)]} \rightarrow \langle * \rangle$$
$$\underline{[f(a)]} \rightarrow [b] \quad \underline{[f(a)]} \rightarrow \langle f(*) \rangle \quad \underline{[g(a,b)]} \rightarrow \langle a \rangle$$
$$\underline{[f(a)]} \rightarrow \langle a \rangle \quad \underline{[f(a)]} \rightarrow [g(a,b)] \quad \underline{[g(a,b)]} \rightarrow [g(a,b)]$$

Again, the automaton $\mathcal{B}_\uparrow$ is similar to $\mathcal{A}_\uparrow$ but with every underlined state replaced by the corresponding non-underlined state and vice-versa. We have $(g(b,b), b) \in \mathcal{L}(\mathcal{G}_\uparrow(\mathcal{R}))$ as $g(b,b) \rightarrow^*_{\mathcal{A}_\uparrow} g(\underline{[a]}, \underline{[b]}) \rightarrow^*_{\mathcal{A}_\uparrow} \underline{[b]} \, {}^*_{\mathcal{B}_\uparrow}\!\!\leftarrow b$.

5 GTT Language Inclusion

The traditional method [5, 6] to decide $\mathcal{L}(\mathcal{G}_1) \subseteq \mathcal{L}(\mathcal{G}_2)$ for GTTs $\mathcal{G}_1$ and $\mathcal{G}_2$ involves an injective transformation $\otimes$ from binary relations on ground terms into sets of ground terms over an extended signature such that $\otimes(R)$ is regular whenever R is a GTT relation.

Let R be an arbitrary binary relation on ground terms in $\mathcal{T}(\mathcal{F})$. Let $\otimes$ be a fresh binary function symbol. The set $\otimes(R)$ consists of the (unique) normal forms of the terms $\otimes(s, t)$ for every pair $(s, t) \in R$ with respect to the TRS consisting of all rewrite rules

$$\otimes(f(\vec{x}), f(\vec{y})) \rightarrow f(\otimes(x_1, y_1), \ldots, \otimes(x_n, y_n))$$

with f an n-ary function symbol different from $\otimes$ and pairwise distinct variables $x_1, \ldots, x_n, y_1, \ldots, y_n$. These rules extract the common context of two terms.

It is easy to see that $\otimes$ is effective and injective. For the proof that $\otimes$ transforms GTT relations into regular sets we refer to [5]. We have $\mathcal{L}(\mathcal{G}_1) \subseteq \mathcal{L}(\mathcal{G}_2)$ if and only if $\otimes(\mathcal{L}(\mathcal{G}_1)) \subseteq \otimes(\mathcal{L}(\mathcal{G}_2))$. Since the latter is decidable, GTT language inclusion is decidable.

We formalized a different method that orginates in the decision procedure for the first-order theory of rewriting for ground TRSs [7], and is implemented in FORT [19]. The central concept in this procedure, besides GTT, is that of an RR_n relation, which are regular tree languages that encodes n-tuples of ground terms.

Definition 5.1. Let $t_1, \ldots, t_n$ be a sequence of ground terms. We define a new ground term $\langle t_1, \ldots, t_n \rangle$ ✔ by taking as the set of positions the union of the sets of positions of $t_1, \ldots, t_n$, and defining the function symbol at position p by $t_1(p) \ldots t_n(n)$, where $t(p)$ is the function symbol at position p of t, or $\perp$ if p is not a position of t.

An RR_n relation is an n-ary relation R on ground terms such that the set

$$\{\langle t_1, \ldots, t_n \rangle \mid (t_1, \ldots, t_n) \in R\}$$

is a regular tree language.

Example 5.2. Considering the signature in Example 3.3, we have $\langle g(f(b), a), g(g(a, b), b) \rangle = gg(fg(ba, \perp b), ab)$.

Remark 5.1. In order for the encoding to yield terms, we cannot allow arbitrary sets of positions. Rather, we are working with so-called tree domains; sets of positions that are closed under taking prefixes and under replacing a position $p.(i + 1)$ by $p.i$. These conditions would be awkward to formalize, so instead we model tree domains as ground terms with only one function symbol, from which the positions can be computed on demand. ✔

Theorem 5.3. *GTT relations are RR_2 relations.* ✔

Proof. We split the construction into two steps. First we show that given a GTT $(\mathcal{A}, \mathcal{B})$, the relation $\mathcal{G}_\epsilon = \{(s, t) \mid s \in \mathcal{L}(\mathcal{A}, q)$ and $t \in \mathcal{L}(\mathcal{B}, q)$ for some state $q\}$ is an RR_2 relation. In a second step, we show that the parallel closure of an RR_2 relation is again an RR_2 relation.

For the first step, 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

$$fg(p_1 q_1, \ldots, p_k q_k) \rightarrow pq$$
$$f\perp(p_1 \perp, \ldots, p_n \perp) \rightarrow p\perp$$
$$\perp g(\perp q_1, \ldots, \perp q_m) \rightarrow \perp q$$

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

$$pq \rightarrow p'q \quad \text{for all } p \rightarrow p' \in \mathcal{A} \text{ and } q \in Q_\mathcal{B} \cup \{\perp\}$$
$$pq \rightarrow pq' \quad \text{for all } q \rightarrow q' \in \mathcal{B} \text{ and } p \in Q_\mathcal{A} \cup \{\perp\} \quad ✔$$

These transitions accept $\langle s, t \rangle$ in state pq if and only if $s \in \mathcal{L}(\mathcal{A}, p)$ and $t \in \mathcal{L}(\mathcal{B}, q)$. As final states we pick pp with $p \in Q_\mathcal{A} \cup Q_\mathcal{B}$. The resulting automaton accepts $\mathcal{G}_\epsilon$ as an RR$_2$ relation. ✔✔✔

The second step depends on the signature $\mathcal{F}$. We take the automaton C from the previous step, and add an extra state $*$ with transitions

$$\{f f(*, \ldots, *) \to * \mid f \in \mathcal{F}\} \cup \{q \to * \mid q \in F_C\} \quad ✔$$

The resulting automaton accepts the parallel closure of C. Composing the two steps concludes the proof. ✔

Based on this construction we can check $\mathcal{L}(\mathcal{G}_1) \subseteq \mathcal{L}(\mathcal{G}_2)$ using a language inclusion check for tree automata, which already exists as part of IsaFoR.

Corollary 5.4. *Ground confluence of LV-TRSs is decidable.*

Proof. Let $\mathcal{R}$ be an LV-TRS. By Theorems 3.2, 4.3 and 4.9, and closure of GTT relations under inverse, both

$$\uparrow_\mathcal{R} = \twoheadrightarrow_\mathcal{R}^{*-} \cdot \twoheadrightarrow_\mathcal{R}^*$$

and

$$\downarrow_\mathcal{R} = \twoheadrightarrow_\mathcal{R}^* \cdot \twoheadrightarrow_\mathcal{R}^{*-}$$

are GTT languages. Using Theorem 5.3, we obtain corresponding RR$_2$ relations. The proofs are based on effective constructions (see also Section 7). We conclude because language inclusion for regular tree languages is decidable. □

6 Confluence vs Ground Confluence

Since tree automata and GTTs operate on ground terms, the formalized results of Sections 3–5 show the decidability of ground confluence for LV-TRSs. In general, ground confluence is not equivalent to confluence. In [20, 21] two sufficient conditions are presented for the equivalence of ground confluence and confluence.

Lemma 6.1. *A ground-confluent TRS $\mathcal{R}$ over a signature $\mathcal{F}$ is confluent if $\mathcal{R}$ is ground or $\mathcal{F}$ is monadic.*

Here monadic means that there are no function symbols of arity two or higher. We formalized the proofs presented in [20]. ✔✔

Confluence is equivalent to ground-confluence for left-linear, right-ground TRSs after a fresh constant is added to the signature [20, 21]. We slightly extended and formalized this result, opening the way for a verified confluence tool for right-ground LV-TRSs.

Theorem 6.2. *Let $\mathcal{R}$ be a right-ground LV-TRS over a signature $\mathcal{F}$ and let $c \notin \mathcal{F}$ be a constant. Then $\mathcal{R}$ is confluent on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ if and only if $\mathcal{R}$ is ground confluent on $\mathcal{T}(\mathcal{F} \cup \{c\})$.* ✔

Below we present a sketch of the formalized proof.

Proof sketch. We write $\mathcal{F}_c$ for $\mathcal{F} \cup \{c\}$. For the *if* direction, we show that confluence is preserved under signature extension. Consider the substitution σ_c that maps every variable to c. To show confluence, it suffices to show existence of a map φ from $\mathcal{T}(\mathcal{F}_c)$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $t = \varphi(t)\sigma_c$ and φ preserves reachability. ✔✔✔ Such a map φ can be constructed by replacing every occurrence of c in a ground term with a fixed variable x. ✔

For the *only if* direction we closely follow the proof in [20]:

- if $t\sigma \to_\mathcal{R}^* u$ then $t \to_\mathcal{R}^* u'$ and $u'\sigma_c = u$ for some term $u' \in \mathcal{T}(\mathcal{F}, \mathcal{V})$,
- if $t \to_\mathcal{R}^* u$ then $u(p) = t(p)$ for all $p \in \mathcal{P}os_\mathcal{V}(u)$.

for all terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. The second property relies on $\mathcal{R}$ being right-ground. ✔

Remark 6.1. The proof of the *if* direction in [20] relies on the fact that confluence is preserved under signature extension. Signature extension is a special case of modularity (Toyama [23]), a celebrated result in term rewriting which has recently been formalized in the context of the more general layer framework [12]. Because we deal with LV-TRSs whose left-hand sides may be variables we cannot reuse the formalization and hence we opted for a simple direct proof.

The following example shows that adding one extra constant is not enough to prove confluence of an LV-TRS by ground confluence.

Example 6.3. Consider the LV-TRS $\mathcal{R} = \{a \to x\}$ over the signature $\mathcal{F} = \{a\}$. It is not confluent, because $x \ _\mathcal{R}\!\leftarrow a \to_\mathcal{R} y$ with distinct variables x and y is a non-joinable peak. However, it is ground-confluent ($a \to_\mathcal{R} a$ is the only possible rewrite step), even after adding a single fresh constant b ($a \to_\mathcal{R} a$ and $a \to_\mathcal{R} b$ are the only rewrite steps). Ground confluence is destroyed after adding a second fresh constant c, which results in the non-joinable peak b $_\mathcal{R}\!\leftarrow a \to_\mathcal{R}$ c.

The following new result shows that two fresh constants suffice to reduce confluence to ground-confluence for LV-TRSs.

Theorem 6.4. *Let $\mathcal{R}$ be an LV-TRS over a signature $\mathcal{F}$ and let $c, d \notin \mathcal{F}$ be constants. Then $\mathcal{R}$ is confluent on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ if and only if $\mathcal{R}$ is ground confluent on $\mathcal{T}(\mathcal{F} \cup \{c, d\})$.*

Proof. We write $\mathcal{F}_{cd}$ for $\mathcal{F} \cup \{c, d\}$. We first prove the *only if* direction, so assume that $\mathcal{R}$ is confluent on $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Consider $s \leftrightarrow_\mathcal{R}^* t$ over $\mathcal{T}(\mathcal{F}_{cd})$. We need to show $s \downarrow_\mathcal{R} t$ over $\mathcal{T}(\mathcal{F}_{cd})$. Let $x, y \in \mathcal{V}$ be distinct variables. We uniformly replace c by x and d by y to obtain a conversion $s' \leftrightarrow_\mathcal{R}^* t'$ over $\mathcal{T}(\mathcal{F}, \mathcal{V})$, noting that these constants can only appear in the substitution or context of a rewrite step, never in the left-hand or right-hand side of the applied rewrite rules. By assumption, s' and t' are joinable over $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Because rewriting is closed under substitutions, this implies

$s = s'\sigma \downarrow_{\mathcal{R}} t'\sigma = t$ over $\mathcal{T}(\mathcal{F}_{cd})$, where σ maps x to c and every other variable to d.

For the *if* direction, assume that $\mathcal{R}$ is confluent on $\mathcal{T}(\mathcal{F}_{cd})$ and consider $s \leftrightarrow^*_{\mathcal{R}} t$ over $\mathcal{T}(\mathcal{F}, \mathcal{V})$. We may assume without loss of generality that this conversion has a root step from left to right. In that case, there is a rule $\ell \to r \in \mathcal{R}$ and a substitution σ with $s \leftrightarrow^*_{\mathcal{R}} \ell\sigma \to_{\mathcal{R}} r\sigma \leftrightarrow^*_{\mathcal{R}} t$. Let $\sigma_c(x) = c$ and $\sigma_d(x) = d$ be substitutions mapping all variables to c and d respectively. Because the variables of ℓ and r are disjoint, there is a rewrite step $\ell\sigma_c \to r\sigma_d$, and hence there exists a conversion $s\sigma_c \leftrightarrow^*_{\mathcal{R}} t\sigma_d$ over $\mathcal{T}(\mathcal{F}_{cd})$. By assumption, $s\sigma_c \downarrow_{\mathcal{R}} t\sigma_d$ over $\mathcal{T}(\mathcal{F}_{cd})$. We claim that this implies $s \to^*_{\mathcal{R}} \cdot {}^*_{\mathcal{R}}{\leftarrow} t$. If there is no root step in $s\sigma_c \to^*_{\mathcal{R}} \cdot {}^*_{\mathcal{R}}{\leftarrow} t\sigma_d$, then neither s nor t can be a variable (since $c, d \notin \mathcal{F}$) so the root symbols of s and t must be the same function symbol, and we can recurse to the arguments. If there is a root step $\ell'\sigma' \to_{\mathcal{R}} r'\sigma'$, we may assume without loss of generality that it is in the left part of the valley, i.e.,

$$s\sigma_c \xrightarrow[\mathcal{R}]{*} \ell'\sigma' \xrightarrow[\mathcal{R}]{} r'\sigma' \xrightarrow[\mathcal{R}]{*} \cdot \xleftarrow[\mathcal{R}]{*} t\sigma_d \tag{1}$$

We claim that we can transform (1) into a valley

$$s \xrightarrow[\mathcal{R}]{*} \ell'\tau \xrightarrow[\mathcal{R}]{} r'\tau \xrightarrow[\mathcal{R}]{*} \cdot \xleftarrow[\mathcal{R}]{*} t \tag{2}$$

over $\mathcal{T}(\mathcal{F}, \mathcal{V})$, using the same rules at the same positions. To convert $s\sigma_c = s_0 \to_{\mathcal{R}} s_1 \to^*_{\mathcal{R}} s_n = \ell'\sigma'$, we start from the left, letting $s'_0 = s$. For each rewrite step $s_i = C[\ell''\sigma'']_p \to C[r''\sigma'']_p = s_{i+1}$ define $\tau'(x)$ for $x \in Var(\ell'')$ by matching $s'_i|_p$ against ℓ'' and $\tau'(x)$ for $x \in Var(r'')$ by the result of replacing the constants c and d in $\sigma''(x)$ by a fixed variable z. Because $Var(\ell'') \cap Var(r'') = \varnothing$, τ' is well-defined. We obtain a rewrite step $s'_i = s'_i[\ell''\tau']_p \to_{\mathcal{R}} s'_i[r''\tau']_p$ and define $s'_{i+1} = s_i[r''\tau']_p$. This takes care of the initial part $s\sigma_c \to^*_{\mathcal{R}} \ell'\sigma'$ of the valley (1). The part $r'\sigma' \to^*_{\mathcal{R}} \cdot {}^*_{\mathcal{R}}{\leftarrow} t\sigma_d$ can be written as $r'\sigma' = t_m {}_{\mathcal{R}^-}{}^*{\leftarrow} \cdot {}^*_{\mathcal{R}}{\leftarrow} t_0 = t\sigma_d$, noting that $\mathcal{R}^-$ is an LV-TRS. This sequence is treated like $s\sigma_c \to^*_{\mathcal{R}} \ell'\sigma'$ before, starting from the right, obtaining terms $t'_0, \ldots, t'_m$. Finally the substitution τ is obtained by matching s'_n against ℓ' and t'_m against r', which succeeds because of linearity and variable-separation. This yields the desired valley (2):

$$s = s'_0 \xrightarrow[\mathcal{R}]{*} s'_n = \ell'\tau \to r'\tau = t'_m \xleftarrow[\mathcal{R}^-]{*} \cdot \xleftarrow[\mathcal{R}]{*} t'_0 = t \qquad \square$$

7 Verified Ground Confluence Checker

We rely on Isabelle/HOL's code generation mechanism [15] to obtain executable code from our formalization. However, most of our definitions cannot be used directly for code export. We rely on the Automatic Data Refinement framework [17] to semi-automatically convert our definitions in terms of sets to definitions that use concrete data types (mostly red-black trees and lists) to represent them.

As a simple example, we have implemented a function that maps the states of an automaton to consecutive natural numbers. The definition is given in Listing 1. This method

```
ta_nat 𝒜 :                                              ✔
    qs ← op_set_to_list (ta_states 𝒜)
    let qm = (map_of (zip qs [0..<length qs]))
    return (fmap_states_ta (the ∘ qm) 𝒜)
```

Listing 1. Abstract implementation of `ta_nat`

obtains the states as a distinct list of states in an unspecified order using the `op_set_to_list` operation, and then renames those states to 0, 1, etc. in the order that they appear in that list.

The concrete definition is obtained by automatic refinement. We specify that we want to refine the tree automata to their default implementation type provided by IsaFoR, and we leave the actual function definition open using a schematic variable (indicated by a question mark, thus: $?f$).   ✔

The concrete implementation is inferred from the abstract implementation by taking the concrete implementation types into account. For example, the concrete datatype for the set of states of the input automaton will be a red-black tree, and the automatic refinement framework has a lemma stating that for red-black trees, so `op_set_to_list` can be implemented by a function returning the list of keys of the tree.

This is just a glimpse of how automatic refinement works, and is a fairly straightforward case. Even in this simple case we had to provide some auxiliary lemmas in order to get an implementation of `map_of`. (This is the reason why we talk of *semi*-automatic refinement above.)

We now change focus to our treatment of inductively defined sets as in Figure 1. We currently have two inductively defined sets in our formalization, and we anticipate that there will be more, for example in the formalization of the *finiteness* predicate Fin provided by FORT [19].

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

$$\frac{\alpha_1 \wedge \cdots \wedge \alpha_n \to \beta \in \mathcal{H} \qquad \overline{\mathcal{H}}(\alpha_i) \text{ for } 1 \leqslant i \leqslant n}{\overline{\mathcal{H}}(\beta)}$$

```
saturate_rec(α, I):                             ✔
    if α ∈ I then
        return I
    else
        J ← {α} ∪ I
        for all β ∈ H₁(α, I) do
            J ← saturate_rec(β, J)
        return J

saturate:                                       ✔
    I ← ∅
    for all α ∈ H₀ do
        I ← saturate_rec(α, I)
    return I
```

Listing 2. Implementing positive Horn inference

Example 7.2. Consider the inference rules from Figure 1. 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
- $(p_1, q_1) \wedge \cdots \wedge (p_n, q_n) \rightarrow (p, q)$ if $f(p_1, \ldots, p_n) \rightarrow_{\mathcal{A}} p$ and $f(q_1, \ldots, q_n) \rightarrow_{\mathcal{B}} q$.

These Horn clauses correspond directly to Figure 1 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

$$\mathcal{H}_0 = \{\beta \mid \top \rightarrow \beta \in \mathcal{H}\}$$ ✔
$$\mathcal{H}_1(\alpha, B) = \{\beta \mid \alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta \in \mathcal{H} \text{ and}$$
$$\alpha \in \{\alpha_1, \ldots, \alpha_n\} \subseteq B \cup \{\alpha\}\}$$ ✔

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 as in Listing 2. 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 red-black trees.   ✔ ✔ ✔

Example 7.3. Continuing from Example 7.2, 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 Figure 1, we obtain the following descriptions:

$$\mathcal{H}_0 = \{(p, q) \mid f() \rightarrow_{\mathcal{A}} p \wedge f() \rightarrow_{\mathcal{B}} q\}$$
$$\mathcal{H}_1((p, q), B) = \{(r, q) \mid p \rightarrow_{\mathcal{A}} r\} \cup \{(p, r) \mid q \rightarrow_{\mathcal{B}} r\} \cup \mathcal{H}_1'$$

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

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

with $(p_i, q_i) \in B \cup \{(p, q)\}$ for all $1 \leqslant i \leqslant n$, and $(p, q) = (p_i, q_i)$ for some $1 \leqslant i \leqslant n$.   ✔ ✔

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.

Remark 7.1. Isabelle/HOL has a *predicate compiler* [2] 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 Figure 1, 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.

Using these building blocks, we provide an abstract implementation of the ground confluence check for LV-TRSs ✔, and prove that it is correct ✔. From this we obtain a refinement working on concrete data types ✔. Based on this implementation, we provide a method for checking ground confluence of LV-TRSs called

```
lv_gcr_procedure_wrap                           ✔
```

which does input validation, i.e., it checks whether the input is an LV-TRS. The function returns MAYBE if input validation fails, and YES or NO otherwise. We also define a *confluence* check for left-linear, right-ground TRSs,

```
llrg_cr_procedure_wrap                          ✔
```

incorporating Theorem 6.2 from the previous section.

Theorem 7.4. *The procedures* lv_gcr_procedure_wrap *and* llrg_cr_proecedure_wrap *are sound, that is:*

- *if* lv_gcr_procedure_wrap $\mathcal{F}$ $\mathcal{R}$ *returns* YES *(*NO*), then* $\mathcal{R}$ *is (non-)ground-confluent over* $\mathcal{F}$,   ✔
- *if* llrg_cr_procedure_wrap $\mathcal{F}$ $\mathcal{R}$ *returns* YES *(*NO*), then* $\mathcal{R}$ *is (non-)confluent over* $\mathcal{F}$.   ✔
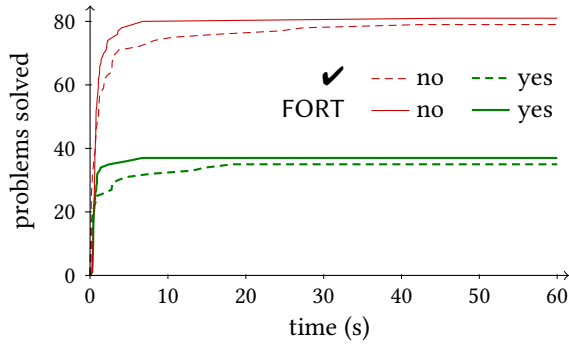
tool	yes	no	timeout		total
✔	35	79	7	(60s)	121
	36	79	6	(600s)	121
FORT	37	81	3	(60s)	121
	38	83	0	(600s)	121

Figure 3. Results for left-linear, right-ground COPS

We have not yet formalized completeness, though on paper we know that the functions terminate and return YES or NO for all valid inputs, i.e., for all LV-TRSs in the case of `lv_gcr_procedure_wrap`, and for left-linear, right-ground TRSs in the case of `llrg_cr_procedure_wrap`. It is worth noting that the Isabelle/HOL code export does not guarantee termination, so having a formalized completeness proof is less valuable than it may seem.

Remark 7.2. Theorem 7.4 marks the boundary between verified and trusted code in our formalization. In order to obtain an actual tool, we export the two functions as Haskell code. The untrusted code is a simple wrapper that uses the WST format parser from the `haskell-rewriting` library [10] for parsing TRSs, and does a straightforward conversion from strings to lists of natural numbers before passing the TRS to the verified code.[3]

8 Assessment

Our ultimate goal is to verify outputs of FORT for the full first-order theory of rewriting by replaying the underlying automata constructions using verified code, so the performance of these constructions is important. We have tested[4] the ground confluence decision procedure on the 121 left-linear, right-ground COPS.[5] In Figure 3 we compare the runtimes of the formalized version (indicated by ✔) to those of FORT. The formalized version is slower. There are a number of reasons why the speed differs, for example:

---

[3]For example, the ground confluence tool wrapper can be found at [Ꮯ]/fortissimo/cpp2019/code/GCR_LV.html.

[4]We used an i7-5930K (12 hyperthreads, 3.5GHz) CPU and 32GB of RAM.

[5]COnfluence ProblemS, http://project-coco.uibk.ac.at/problems/
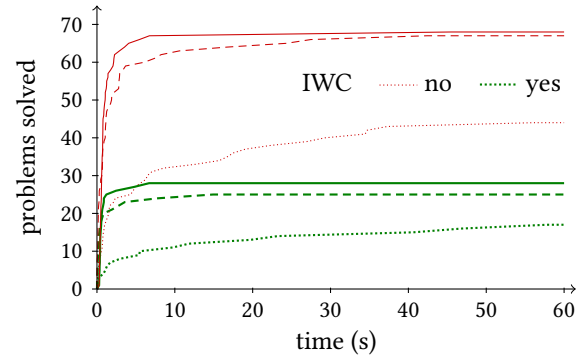
tool	yes	no	timeout		total
✔	25	67	6	(60s)	98
	26	67	5	(600s)	98
IWC	17	45	36	(60s)	98
	22	54	22	(600s)	98
FORT	28	68	2	(60s)	98
	29	69	0	(600s)	98
CSI	29	69	0	(1s)	98

Figure 4. Experimental results for ground COPS

- The tree automata language inclusion check as formalized in IsaFoR needlessly does a subset construction for both automata. Since $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ can be reformulated as $\mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathcal{B})} = \varnothing$, no subset construction for the first automaton is required.

- FORT uses *semi-confluence* ($\leftarrow \cdot \rightarrow^* \subseteq \downarrow$) for checking confluence, resulting in smaller GTTs for the peak.

- FORT does not implement GTT composition, but instead relies on the RR_n machinery to express joinability.

- The implementation language is different, Java for FORT, and Haskell for the formalized code.

In Figure 4 we compare our current implementation to the preliminary version presented at IWC 2018 [11]. That version was restricted to ground TRSs, so in order to make the comparison fair, we restrict the data set to the 98 ground COPS. We also include FORT and CSI in the comparison. Note that CSI implements a cubic time decision procedure [9] for (ground) confluence of ground TRSs, with negligible runtime on all ground Cops; we omit it from the graph for that reason. The new formalized version is much faster than the one presented at IWC. The main reason for this is the incorporation of trimming before the conversion of GTTs to RR_2 automata.

topic	LoI
Fundamentals	933
GCR/CR	1483
GTT/RR$_2$	4093
Execution	2257
total	8766

Figure 5. Size of formalization

On the accompanying website we also provide results for confluence.[6] These are not very different from the results for ground confluence; the extra constant required by Theorem 6.2 has almost no effect on the runtime except when a system is non-confluent, which allows the language inclusion check to terminate early.

An overview of the size of the formalization (measured in Lines of Isar) is given in Figure 5. We categorized the theories roughly by purpose.

- **Fundamentals**: This covers some preliminaries not covered in IsaFoR, including definitions of ground terms, GTTs and GTT relations, and some miscellaneous auxiliary facts.

- **GCR/CR**: This material concerns the relation between ground confluence and confluence. See also Section 6.

- **GTT/RR$_2$**: Here we prove the TRS to GTT conversion and GTT closure properties from Sections 3 and 4. In addition we also have constructions for cylindrification and projection, which will be important for dealing with the full first-order theory of ground rewriting.

- **Execution**: This covers Horn inference and executable code for all building blocks for the ground confluence decision procedure presented in this paper.

Our formalization is based on IsaFoR (and consequently, on Isabelle/HOL). This is a classical logic, so in contrast to, for example, Coq, we cannot rely on code extraction to obtain executable code; instead we use code export, which uses definitions of functions and datatypes as the basis for producing code in a functional language. This code generator is part of the trusted code base. We believe that aside from reducing the trusted code base, code extraction from proofs is inferior to code generation from explicit definitions, because proofs tend to favor simple arguments that correspond to very naive and slow algorithms. So even in Coq, we would probably end up writing our desired code as a definitions and then do a separate correctness proof.

9 Future Work

There are a number of tasks ahead of us. First of all, we plan to create an AFP[7] entry covering GTTs and the constructions described in this paper, including the direct language inclusion construction from [5]. We will also work on improving the performance of the existing constructions, where language inclusion is the obvious starting point.

Another big task will be covering the full first-order theory of rewriting [7], for which we will need basic first-order logic (which we hope to borrow from an existing formalization) and its connection to projection and cylindrification; furthermore, we need executable code for these constructions.

Last but not least, we plan to formalize Theorem 6.4 in order to obtain a confluence tool that covers LV-TRSs in full generality.

Acknowledgments

References

[1] F. Baader and T. Nipkow. 1998. *Term Rewriting and All That*. Cambridge University Press. https://doi.org/10.1017/CBO9781139172752

[2] S. Berghofer, L. Bulwahn, and F. Haftmann. 2009. Turning Inductive into Equational Specifications. In *Proc. 22nd TPHOLs (LNCS)*, Vol. 5674. 131–146. https://doi.org/10.1007/978-3-642-03359-9_11

[3] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 2008. Tree Automata Techniques and Applications. http://www.grappa.univ-lille3.fr/tata

[4] H. Comon, G. Godoy, and R. Nieuwenhuis. 2001. The Confluence of Ground Term Rewrite Systems is Decidable in Polynomial Time. In *Proc. 42nd FOCS*. 298–307. https://doi.org/10.1109/SFCS.2001.959904

[5] M. Dauchet, T. Heuillard, P. Lescanne, and S. Tison. 1990. Decidability of the Confluence of Finite Ground Term Rewriting Systems and of Other Related Term Rewriting Systems. *I&C* 88, 2 (1990), 187–201. https://doi.org/10.1016/0890-5401(90)90015-A

[6] M. Dauchet and S. Tison. 1985. Decidability of Confluence for Ground Term Rewriting Systems. In *Proc. 5th FCT (LNCS)*, Vol. 199. 80–84. https://doi.org/10.1007/BFb0028794

[7] M. Dauchet and S. Tison. 1990. The Theory of Ground Rewrite Systems is Decidable. In *Proc. 5th LICS*. 242–248. https://doi.org/10.1109/LICS.1990.113750

[8] M. Dauchet, S. Tison, T. Heuillard, and P. Lescanne. 1987. Decidability of the Confluence of Ground Term Rewriting Systems. In *Proc. 2nd LICS*. 353–359.

[9] B. Felgenhauer. 2012. Deciding Confluence of Ground Term Rewrite Systems in Cubic Time. In *Proc. 23rd RTA (LIPIcs)*, Vol. 15. 165–175. https://doi.org/10.4230/LIPIcs.RTA.2012.165

[10] B. Felgenhauer, M. Avanzini, and C. Sternagel. 2013. A Haskell Library for Term Rewriting. *CoRR* abs/1307.2328 (2013). http://arxiv.org/abs/1307.2328

[11] B. Felgenhauer, A. Middeldorp, T. V. H. Prathamesh, and F. Rapp. 2018. Towards a Verified Decision Procedure for Confluence of Ground Term Rewrite Systems in Isabelle/HOL. In *Proc. 7th IWC*. 46–50. Available at http://cl-informatik.uibk.ac.at/iwc/iwc2018.pdf.

---

[6][ ]/fortissimo/cpp2019/experiments-cr/tools.php

---

[7]Archive of Formal Proofs, https://www.isa-afp.org/

[12] B. Felgenhauer and F. Rapp. 2018. Layer Systems for Confluence – Formalized. In *Proc. 15th ICTAC (LNCS)*, Vol. 11187. 1–19. https://doi.org/10.1007/978-3-030-02508-3_10

[13] G. Godoy and A. Tiwari. 2005. Confluence of Shallow Right-Linear Rewrite Systems. In *Proc. 14th CSL (LNCS)*, Vol. 3634. 541–556. https://doi.org/10.1007/11538363_37

[14] G. Godoy, A. Tiwari, and R. Verma. 2004. Characterizing Confluence by Rewrite Closure and Right Ground Term Rewrite Systems. *AAECC* 15 (2004), 13–36. https://doi.org/10.1007/s00200-004-0148-6

[15] F. Haftmann and T. Nipkow. 2010. Code Generation via Higher-Order Rewrite Systems. In *Proc. 10th FLOPS (LNCS)*, Vol. 6009. 103–117. https://doi.org/10.1007/978-3-642-12251-4_9

[16] L. Kaiser. 2005. Confluence of Right Ground Term Rewriting Systems is Decidable. In *Proc. 8th FoSSaCS (LNCS)*, Vol. 3441. 470–489. https://doi.org/10.1007/978-3-540-31982-5_30

[17] P. Lammich. 2013. Automatic Data Refinement. In *Proc. 4th ITP (LNCS)*, Vol. 7998. 84–99. https://doi.org/10.1007/978-3-642-39634-2_9

[18] M. Oyamaguchi. 1987. *The Church-Rosser Property for Quasi-Ground Term Rewriting Systems*. Technical Report. Faculty of Engineering, Mie University, Japan.

[19] F. Rapp and A. Middeldorp. 2016. Automating the First-Order Theory of Left-Linear Right-Ground Term Rewrite Systems. In *Proc. 1st FSCD (LIPIcs)*, Vol. 52. 36:1–36:12. https://doi.org/10.4230/LIPIcs.FSCD.2016.36

[20] F. Rapp and A. Middeldorp. 2016. Confluence Properties on Open Terms in the First-Order Theory of Rewriting. In *Proc. 5th IWC*. 26–30. Available at http://cl-informatik.uibk.ac.at/iwc/iwc2016.pdf.

[21] F. Rapp and A. Middeldorp. 2018. FORT 2.0. In *Proc. 9th IJCAR (LNAI)*, Vol. 10900. 81–88. https://doi.org/10.1007/978-3-319-94205-6_6

[22] R. Thiemann and C. Sternagel. 2009. Certification of Termination Proofs using CeTA. In *Proc. 22nd TPHOLs (LNCS)*, Vol. 5674. 452–468. https://doi.org/10.1007/978-3-642-03359-9_31

[23] Y. Toyama. 1987. On the Church-Rosser Property for the Direct Sum of Term Rewriting Systems. *J. ACM* 34, 1 (1987), 128–143. https://doi.org/10.1145/7531.7534