Information and Computation 253 (2017) 467–483

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Reachability, confluence, and termination analysis with state-compatible automata ☆

Bertram Felgenhauer *, René Thiemann

Institute of Computer Science, University of Innsbruck, Innsbruck, Austria

ARTICLE INFO

Article history:
Received 28 June 2014
Available online 6 June 2016

ABSTRACT

Regular tree languages are a popular device for reachability analysis over term rewrite systems, with many applications like analysis of cryptographic protocols, or confluence and termination analysis. At the heart of this approach lies tree automata completion, first introduced by Genet for left-linear rewrite systems. Korp and Middeldorp introduced so-called quasi-deterministic automata to extend the technique to non-left-linear systems. In this paper, we introduce the simpler notion of state-compatible automata, which are slightly more general than quasi-deterministic, compatible automata. This notion also allows us to decide whether a regular tree language is closed under rewriting, a problem which was not known to be decidable before.

The improved precision has a positive impact in applications which are based on reachability analysis, namely termination and confluence analysis.

Our results have been formalized in the theorem prover Isabelle/HOL. This allows to certify automatically generated proofs that are using tree automata techniques.

© 2016 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

In this paper we are largely concerned with over-approximations of the terms reachable from a regular tree language L_0 by rewriting using a term rewrite system $\mathcal{R}$, that is, we are interested in regular tree languages L such that $\mathcal{R}^*(L_0) \subseteq L$. Such over-approximations have been used, among other things, in the analysis of cryptographic protocols [8], for termination analysis [9,13] and for establishing non-confluence of term rewrite systems [19]. These applications work by first translating a problem into a term rewrite system: states (of, say, a program or a protocol) are modeled as terms, and behaviors (the effect of program statements, or of actions by the protocol participants and attackers) are modeled by rules of a term rewrite system. In the context of term rewriting, the most basic use of over-approximating reachable terms is for non-reachability: Given a set of starting terms L_0, a term rewrite system $\mathcal{R}$, and a set of bad terms B, are there terms $s \in L_0$ and $b \in B$ such that s reaches b? If we can find an over-approximation L of $\mathcal{R}^*(L_0)$ such that $L \cap B$ is non-empty, then we can answer this question negatively. Note that it is beneficial to make the approximation L as small as possible.

Unfortunately, the question whether $\mathcal{R}^*(L_0) \subseteq L$ for regular languages L_0 and L and a term rewrite system $\mathcal{R}$ is undecidable in general. Tree automata completion, conceived by Genet et al. [6,7], is based on the stronger requirements that $L_0 \subseteq L$ and L is itself closed under rewriting, i.e., $\mathcal{R}(L) \subseteq L$.

☆ This research was supported by FWF projects P22467, P22767, P27528 and Y757.

* Corresponding author.
E-mail addresses: bertram.felgenhauer@uibk.ac.at (B. Felgenhauer), rene.thiemann@uibk.ac.at (R. Thiemann).

http://dx.doi.org/10.1016/j.ic.2016.06.011
0890-5401/© 2016 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

In the present paper we concentrate on the second property, $\mathcal{R}(L) \subseteq L$, since checking $L_0 \subseteq L$ for regular languages L_0 and L is a well-studied problem. This research was born of involvement in the development of three tools for term rewriting, CeTA [18], a certifier for termination and confluence proofs generated by provers, CSI [19], an automated confluence prover, and $\mathsf{T_TT_2}$ [11], an automated termination prover. The general idea is that the automated provers generate proofs of (non-)confluence or (non-)termination of a term rewrite system, which can then be verified by a certifier like CeTA. The distinguishing feature of a certifier is that it is highly trustworthy; in the case of CeTA, this is achieved by proving its code correct in the proof assistant Isabelle [17]. In contrast, CSI and $\mathsf{T_TT_2}$ are ordinary programs that are far more likely to contain bugs. Both CSI and $\mathsf{T_TT_2}$ use quasi-deterministic automata [13] to produce overapproximations of reachable terms, and CeTA could not certify the resulting proofs. Our main contributions are as follows:

State-compatible automata We introduce the notion of state-compatible automata as a way of ensuring closure under rewriting. Our definition refines Genet's concept of compatibility [7], allowing better (i.e., smaller) approximations of reachable terms in some cases. At the same time, our notion is considerably simpler than quasi-deterministic automata, which is beneficial for the formalization in Isabelle. We show that state-compatibility does not only ensure $\mathcal{R}(L) \subseteq L$, but can also be utilized to obtain a decision procedure for the question whether a regular tree language is closed under rewriting—to the best of our knowledge, this decidability result is new. The theory of state-compatible automata is developed in Section 3.

Comparison to quasi-deterministic automata We carefully analyze how state-compatible automata relate to quasi-deterministic automata, which were introduced by Korp and Middeldorp [13] to overcome problems with tree automata completion for non-left-linear term rewrite systems. We show that every quasi-deterministic automaton can easily be converted to a state-compatible automaton accepting the same language. This conversion is currently used in CSI (for non-confluence) and $\mathsf{T_TT_2}$ (for termination) in order to produce certifiable proofs when tree automata are used. The relation to quasi-deterministic automata can be found in Section 4.1, and we relate state-compatible automata to quasi-models by Endrullis et al. [4] in Section 4.2.

Adaptation to match-bounds The match-bounds technique by Geser et al. [9] is a technique for proving termination of term rewrite systems. The original technique is a direct application of tree automata completion, but is restricted to left-linear term rewrite systems. Korp and Middeldorp [13] extended match-bounds to non-left-linear systems, using so-called raise-consistent, quasi-deterministic automata. We show how to adapt raise-consistency for state-compatible automata in Section 6.

Formalization Last but not least, we formalized state-compatible automata and the applications to non-confluence and termination in Isabelle, making the techniques available in the certifier CeTA, and allowing it to certify more non-confluence and termination proofs produced by CSI and $\mathsf{T_TT_2}$. We also formalized the completeness result for checking $\mathcal{R}(L) \subseteq L$, so that CeTA can, in principle, verify match-bounds and non-confluence proofs by tools that are unaware of state-compatible automata. See Section 7 for details.

Our formalization improves on the certifier for tree automata completion by Boyer et al. [2] in that we can handle non-left-linear systems.

Furthermore, state-compatible automata strictly refine both compatible and quasi-compatible automata, which matters for applications. To demonstrate this, we provide examples where the techniques of [12,13,19] are successfully applied when state-compatible automata are used, but where the techniques must fail if they are restricted to use the compatibility criteria of Genet or Korp and Middeldorp. These examples can be found in Sections 5 and 6. Finally, we conclude in Section 8.

Remark A preliminary version of this paper was already published in [5]. The current paper extends this work in various directions. For example, whereas [5] only mentions the application areas in passing, we now provide many more details in Sections 5 and 6. These include the new Examples 28 and 43 which show that the improved precision of state-compatible automata carries over to the applications. Even more importantly, we now present all details on how to adapt state-compatible automata for match-bounds to non-left-linear term rewrite systems, a task which was only mentioned as future work in [5]. The necessary results of Sections 6.1 and 6.2 have also been formalized using Isabelle. We also provide some insight into the implementation of tree automata completion in $\mathsf{T_TT_2}$, which is based on an unpublished extension of quasi-deterministic automata with ε-transitions by Korp. Finally, Section 7 on our formalization has been extended.

2. Preliminaries

We assume that the reader is familiar with first order term rewriting and tree automata. For introductions to these topics see [1] and [3].

Terms over a signature $\mathcal{F}$ and a set of variables $\mathcal{V}$, denoted $\mathcal{T}(\mathcal{F}, \mathcal{V})$ (or $\mathcal{T}(\mathcal{F})$ if $\mathcal{V}$ is empty) are inductively defined as either variables $v \in \mathcal{V}$ or of the form $f(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms and $f \in \mathcal{F}$ is a function symbol of arity n. We write $\mathcal{V}\mathrm{ar}(t)$ for the set of variables in t. A term t is linear if each variable occurs in t at most once. Contexts are terms over $\mathcal{F} \cup \{\Box\}$ that contain exactly one occurrence of $\Box$. If C is a context and t a term, then $C[t]$ denotes the term obtained by

replacing the $\Box$ in C by t. There are also multi-hole contexts D where $D\langle t_1, \ldots, t_n\rangle$ is the term that is obtained by filling all holes: we always assume that the number n of terms matches the number of holes in D. A substitution $\tau : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ maps variables to terms. We write $t\tau$ for the result of replacing each variable x in t by $\tau(x)$.

A term rewrite system (TRS) $\mathcal{R}$ is a set of rewrite rules $l \to r$, where each rule's left-hand side l and right-hand side r are terms such that $l \notin \mathcal{V}$ and $\mathcal{V}\mathrm{ar}(r) \subseteq \mathcal{V}\mathrm{ar}(l)$. A TRS $\mathcal{R}$ defines a rewrite relation $\to_{\mathcal{R}}$, namely $s \to_{\mathcal{R}} t$ whenever there are a context C, a rule $l \to r \in \mathcal{R}$, and a substitution τ such that $s = C[l\tau]$ and $t = C[r\tau]$. We denote by $\mathrm{lhs}(\mathcal{R})$ the set of all left-hand sides of rules in $\mathcal{R}$. A TRS is left-linear if all its left-hand sides are linear terms. A rule $l \to r$ is called collapsing if r is a variable. The inverse, the reflexive closure, transitive closure, and the reflexive, transitive closure of a binary relation $\to$ are denoted by $\leftarrow$, $\to^=$, $\to^+$, and $\to^*$, respectively. Given a set of terms L, $\mathcal{R}(L)$ ($\mathcal{R}^*(L)$) is the set of one-step (many-step) descendants of L: $t' \in \mathcal{R}(L)$ ($t' \in \mathcal{R}^*(L)$) if and only if $t \to_{\mathcal{R}} t'$ ($t \to^*_{\mathcal{R}} t'$) for some $t \in L$. A language L is closed under rewriting by $\mathcal{R}$ if $\mathcal{R}(L) \subseteq L$.

A (bottom-up) tree automaton $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ over a signature $\mathcal{F}$ consists of a set of states Q disjoint from $\mathcal{F}$, a set of final states $Q_f \subseteq Q$, and a set of transitions Δ of shape $f(q_1, \ldots, q_n) \to q$ where the root $f \in \mathcal{F}$ has arity n and $q, q_1, \ldots, q_n \in Q$. (We forbid ε-transitions for the sake of simplicity.) We regard Δ as a TRS over the signature $\mathcal{F} \cup Q$, with the states as constants. A substitution σ is a state substitution if $\sigma(x) \in Q$ for all $x \in \mathcal{V}$. A term t is accepted in state q if $t \to^*_\Delta q$; t is accepted by $\mathcal{A}$ if it is accepted in a final state. The language accepted by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{t \mid t \to^*_\Delta q \text{ for some } q \in Q_f\}$. A tree automaton is finite if its set of transitions is finite. A language is regular if it is accepted by a finite tree automaton. We call $\mathcal{A}$ deterministic if no two rules in Δ have the same left-hand side. For convenience, we often write $\to_{\mathcal{A}}$ for $\to_\Delta$. Following [13], we formulate Genet's result from [7] as follows:

Definition 1. A tree automaton $\mathcal{A}$ is *compatible* with a TRS $\mathcal{R}$ if for all state substitutions σ, rules $l \to r \in \mathcal{R}$ and states $q \in Q$, $l\sigma \to^*_{\mathcal{A}} q$ implies $r\sigma \to^*_{\mathcal{A}} q$.

Theorem 2. *Let the tree automaton $\mathcal{A}$ be compatible with the TRS $\mathcal{R}$. Then*

1. *if $\mathcal{R}$ is left-linear, then $\mathcal{L}(\mathcal{A})$ is closed under rewriting by $\mathcal{R}$, and*
2. *if $\mathcal{A}$ is deterministic, then $\mathcal{L}(\mathcal{A})$ is closed under rewriting by $\mathcal{R}$.*

Finally, we recall that every tree automaton can be reduced to an equivalent automaton where all states are useful.

Definition 3. Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ be a tree automaton. We say that a state $q \in Q$ is *reachable* if $t \to^*_{\mathcal{A}} q$ for some term $t \in \mathcal{T}(\mathcal{F})$; $q \in Q$ is *productive* if $C[q] \to^*_{\mathcal{A}} q'$ for some context C and state $q' \in Q_f$. Finally, an automaton $\mathcal{A}$ is *trim* if all its states are both reachable and productive.

Proposition 4. *For any tree automaton $\mathcal{A}$ there is an equivalent tree automaton $\mathcal{A}'$ that is trim. If $\mathcal{A}$ is deterministic, then $\mathcal{A}'$ is also deterministic.*

3. State-compatible automata

In this section, we introduce state-compatible automata. But first we will review tree automata completion and the shortcomings of compatible and quasi-deterministic automata.

3.1. Background and motivation

In tree automata completion, closure under rewriting is ensured by constructing L as the language accepted by a tree automaton $\mathcal{A}$ that is compatible with $\mathcal{R}$. However, for non-left-linear systems, compatibility is not sufficient for ensuing closure under rewriting.

Example 5. Let $\mathcal{R} = \{\mathsf{f}(x, x) \to x\}$ and $\mathcal{A}$ be the automaton with states $1, 2, 3$, final state 3, and transitions

$$\mathsf{a} \to 1 \qquad \mathsf{a} \to 2 \qquad \mathsf{f}(1, 2) \to 3$$

So $\mathcal{A}$ is non-deterministic and $\mathcal{R}$ is non-left-linear. Even though $\mathcal{A}$ is compatible with $\mathcal{R}$, $\mathcal{L}(\mathcal{A}) = \{\mathsf{f}(\mathsf{a}, \mathsf{a})\}$ is not closed under rewriting by $\mathcal{R}$, because $\mathsf{f}(\mathsf{a}, \mathsf{a})$ can be rewritten to a which is not in $\mathcal{L}(\mathcal{A})$.

The usual approach to deal with non-left-linear TRSs is to demand that the automaton is deterministic. However, this may result in bad approximations, as demonstrated by the next example.

Example 6. Let $\mathcal{R} = \{\mathsf{f}(x, x) \to \mathsf{b}, \mathsf{b} \to \mathsf{a}\}$ and $L_0 = \{\mathsf{f}(\mathsf{a}, \mathsf{a})\}$. The set of terms reachable from L_0, namely $\mathcal{R}^*(L_0) = \{\mathsf{f}(\mathsf{a}, \mathsf{a}), \mathsf{b}, \mathsf{a}\}$, is not accepted by any deterministic, compatible tree automaton. To see why, assume that such an automaton

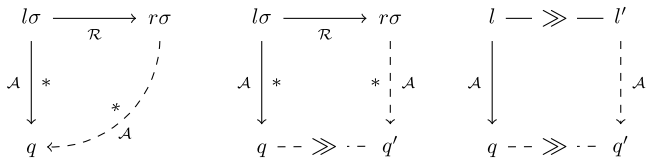

Fig. 1. Compatibility, state-compatibility, and state-coherence.

$\mathcal{A}$ exists, and let q be the state accepting $\mathsf{f}(\mathsf{a},\mathsf{a})$. There must be transitions $\mathsf{a} \to q'$ (q' is unique because $\mathcal{A}$ is deterministic) and $\mathsf{f}(q',q') \to q$ in $\mathcal{A}$. By compatibility with the rules $\mathsf{f}(x,x) \to \mathsf{b}$ and $\mathsf{b} \to \mathsf{a}$, we must have transitions $\mathsf{b} \to q$, and $\mathsf{a} \to q$. Since we already have the transition $\mathsf{a} \to q'$, determinism implies $q' = q$. With the three transitions $\mathsf{a} \to q$, $\mathsf{b} \to q$, and $\mathsf{f}(q,q) \to q$, $\mathcal{A}$ accepts every term over the signature $\{\mathsf{f},\mathsf{a},\mathsf{b}\}$, which is not a very useful approximation of $\mathcal{R}^*(L_0)$.

To overcome this problem, Korp and Middeldorp introduced quasi-deterministic automata [13] (cf. Definition 17). Indeed it is easy to find a quasi-deterministic automaton accepting $\mathcal{R}^*(L_0) = \{\mathsf{f}(\mathsf{a},\mathsf{a}),\mathsf{b},\mathsf{a}\}$ that is compatible with $\mathcal{R}$ from the previous example.

Example 7. Let $\mathcal{A}$ be an automaton with states $1, 2$, final state 2 and transitions

$$\mathsf{a} \to 1^* \qquad \mathsf{a} \to 2 \qquad \mathsf{b} \to 2^* \qquad \mathsf{f}(1,1) \to 2^*$$

(The stars can be ignored for the moment. They indicate the *designated* states for each left-hand side, cf. Definition 17.) Then $\mathcal{A}$ is quasi-deterministic, compatible with $\mathcal{R}$ and $\mathcal{L}(\mathcal{A}) = \{\mathsf{f}(\mathsf{a},\mathsf{a}),\mathsf{b},\mathsf{a}\}$. Hence $\mathcal{L}(\mathcal{A})$ is closed under rewriting by $\mathcal{R}$.

We extend compatible automata in a slightly different way to cover non-left-linear systems. Namely, we stick to deterministic automata, but relax the compatibility restriction to *state-compatibility* instead, which accomplishes a similar effect as quasi-deterministic automata. It turns out that as long as $\mathcal{R}$ has only non-collapsing rules, state-compatible automata and quasi-deterministic automata are equivalent, i.e., they can express the same regular languages that are closed under rewriting by $\mathcal{R}$. In the presence of collapsing rules, state-compatible automata can capture more approximations than quasi-deterministic ones.

3.2. Definitions

Before we get down to definitions, let us briefly analyze the failure in Example 6. What happens there is that, by the compatibility requirement, all three terms in the rewrite sequence $\mathsf{f}(\mathsf{a},\mathsf{a}) \to_{\mathcal{R}} \mathsf{b} \to_{\mathcal{R}} \mathsf{a}$ have to be accepted in the same state. In conjunction with the determinism requirement, this is fatal. Consequently, because our goal is to obtain a deterministic automaton, we must allow a and b to be accepted in separate states, q_{a} and q_{b}. To track their connection by rewriting, we introduce a relation $\gg$ on states, such that $q_{\mathsf{b}} \gg q_{\mathsf{a}}$. In general, we require $\gg$ to be state-compatible and state-coherent, which are defined as follows (see also Fig. 1).

Definition 8. Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ be a tree automaton, and $\gg \subseteq Q \times Q$ be a relation on the states of $\mathcal{A}$. We say that $(\mathcal{A},\gg)$ is *state-compatible with a TRS* $\mathcal{R}$ if for all state substitutions σ, rules $l \to r \in \mathcal{R}$ and states $q \in Q$, if $l\sigma \to^*_{\mathcal{A}} q$ then $r\sigma \to^*_{\mathcal{A}} q'$ for some $q' \in Q$ with $q \gg q'$. We say that $(\mathcal{A},\gg)$ is *state-coherent* if $\{q' \mid q \in Q_f, q \gg q'\} \subseteq Q_f$, and if for all $f(q_1,\dots,q_i,\dots,q_n) \to q \in \Delta$ and $q_i \gg q_i'$ there is some $q' \in Q$ with $f(q_1,\dots,q_i',\dots,q_n) \to q' \in \Delta$ and $q \gg q'$.

The purpose of state-coherence is to deal with contexts in rewrite steps, as we will see in the proof of Theorem 11 below.

Example 9. Let $\mathcal{A}$ be an automaton with states $1, 2$ (both final), and transitions

$$\mathsf{a} \to 1 \qquad \mathsf{b} \to 2 \qquad \mathsf{f}(1,1) \to 2$$

Furthermore, let $2 \gg 2$ and $2 \gg 1$. Then $(\mathcal{A},\gg)$ is state-coherent and state-compatible with $\mathcal{R} = \{\mathsf{f}(x,x) \to \mathsf{b}, \mathsf{b} \to \mathsf{a}\}$ and $\mathcal{L}(\mathcal{A}) = \{\mathsf{f}(\mathsf{a},\mathsf{a}),\mathsf{b},\mathsf{a}\}$. Note that this automaton was obtained from the quasi-deterministic automaton from Example 7 by keeping only the transitions to designated states. We will see in Section 4.1 that this construction works in general.

Remark 10. If $(\mathcal{A},\gg)$ is state-coherent, then $(\mathcal{A},\gg^=)$ and $(\mathcal{A},\gg^*)$ are also state-coherent. The same holds for state-compatibility with $\mathcal{R}$.

3.3. Soundness and completeness

Next we prove the analogue of Theorem 2 for state-coherent, state-compatible automata.

Theorem 11. *Let $\mathcal{A}$ be a tree automaton such that $(\mathcal{A}, \gg)$ is state-coherent and state-compatible with the TRS $\mathcal{R}$ for some relation $\gg$. Then*

1. *if $\mathcal{R}$ is left-linear, then $\mathcal{L}(\mathcal{A})$ is closed under rewriting by $\mathcal{R}$, and*
2. *if $\mathcal{A}$ is deterministic, then $\mathcal{L}(\mathcal{A})$ is closed under rewriting by $\mathcal{R}$.*

Proof. Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$. First we show that whenever $l\tau \to_{\mathcal{A}}^* q$ for some substitution τ and rule $l \to r \in \mathcal{R}$, then there is a state $q' \in Q$ with $q \gg q'$ and $r\tau \to_{\mathcal{A}}^* q'$. By the assumptions, we can extract from a sequence $l\tau \to_{\mathcal{A}}^* q$ a state substitution σ such that $l\tau \to_{\mathcal{A}}^* l\sigma \to_{\mathcal{A}}^* q$: For each $x \in \mathcal{V}\mathrm{ar}(l)$, we map x to the state reached from $\tau(x)$ in the given sequence. The state is unique either by left-linearity, or because the given automaton is deterministic. By state-compatibility, we obtain a state q' such that $q \gg q'$ and $r\tau \to_{\mathcal{A}}^* r\sigma \to_{\mathcal{A}}^* q'$.

Using state-coherence we can show by structural induction on C that whenever $C[q] \to_{\mathcal{A}}^* q_\bullet$ and $q \gg q'$, then $C[q'] \to_{\mathcal{A}}^* q'_\bullet$ for some state $q'_\bullet$ with $q_\bullet \gg q'_\bullet$.

Finally, assume that $t \in \mathcal{L}(\mathcal{A})$ and $t \to_{\mathcal{R}} t'$. Then there exist a rule $l \to r \in \mathcal{R}$, a context C and a substitution τ such that $t = C[l\tau]$ and $t' = C[r\tau]$. We have a derivation $t = C[l\tau] \to_{\mathcal{A}}^* C[q] \to_{\mathcal{A}}^* q_\bullet \in Q_f$. By the preceding observations we can find states $q \gg q'$ and $q_\bullet \gg q'_\bullet$ such that $t' = C[r\tau] \to_{\mathcal{A}}^* C[q'] \to_{\mathcal{A}}^* q'_\bullet$. Note that by state-coherence, $q_\bullet \in Q_f$ implies $q'_\bullet \in Q_f$, so that $t' \in \mathcal{L}(\mathcal{A})$. □

Note that Theorem 11 generalizes Theorem 2 (choose $\gg$ to be the identity relation on states, which is always state-coherent). Moreover, the converse of Theorem 11 holds for trim, deterministic automata. We will prove this in Theorem 13 below, which allows us to derive our main decidability result in Corollary 14. But first let us show by example that the converse fails for some trim, non-deterministic automaton and ground TRS $\mathcal{R}$.

Example 12. Consider the TRS $\mathcal{R} = \{\mathsf{a} \to \mathsf{b}\}$ and the automaton $\mathcal{A}$ with states $0, 1, 2, 3$, final state 0, and transitions

$$\mathsf{a} \to 1 \qquad \mathsf{b} \to 2 \qquad \mathsf{f}(1) \to 0 \qquad \mathsf{g}(1) \to 0$$
$$\mathsf{b} \to 3 \qquad \mathsf{f}(2) \to 0 \qquad \mathsf{g}(3) \to 0$$

This automaton accepts $\mathcal{L}(\mathcal{A}) = \{\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}), \mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{b})\}$, which is closed under rewriting by $\mathcal{R}$. Assume that $(\mathcal{A}, \gg)$ is state-coherent and state-compatible with $\mathcal{R}$. By state-compatibility, $\mathsf{a} \to \mathsf{b}$ begets $1 \gg 2$ or $1 \gg 3$. If $1 \gg 2$, then state-coherence, considering the transition $\mathsf{g}(1) \to 0$, requires a transition with left-hand side $\mathsf{g}(2)$, which does not exist. Similarly, if $1 \gg 3$, then $\mathsf{f}(1) \to 0$ requires a transition with left-hand side $\mathsf{f}(3)$, which does not exist.

Theorem 13. *Let $\mathcal{A}$ be a trim, deterministic tree automaton such that $\mathcal{L}(\mathcal{A})$ is closed under rewriting by the TRS $\mathcal{R}$. Then there is a relation $\gg$ such that $(\mathcal{A}, \gg)$ is state-coherent and state-compatible with $\mathcal{R}$.*

Proof. Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$. We define $\gg$ as follows: $q \gg q'$ if and only if for some terms $t, t' \in \mathcal{T}(\mathcal{F})$, we have

$$q \xleftarrow[\mathcal{A}]{*} t \xrightarrow[\mathcal{R}]{} t' \xrightarrow[\mathcal{A}]{*} q' \tag{1}$$

Note that by virtue of $\mathcal{A}$ being deterministic, t and t' determine q and q' uniquely. We show that $(\mathcal{A}, \gg)$ is state-coherent and state-compatible.

1. (state-coherence) If $q \in Q_f$ and $q \gg q'$, then there exist terms t, t' satisfying (1). In particular, $q \in Q_f$ implies $t \in \mathcal{L}(\mathcal{A})$, and $t \to_{\mathcal{R}} t'$ implies $t' \in \mathcal{L}(\mathcal{A})$, because $\mathcal{L}(\mathcal{A})$ is closed under rewriting by $\mathcal{R}$. Because $\mathcal{A}$ is deterministic, t' determines q' uniquely, and $q' \in Q_f$ follows.
2. (state-coherence) Assume that $f(q_1, \ldots, q_n) \to q \in \Delta$ and $q_i \gg q'_i$ for some index i and state q'_i. By (1) there are t_i, t'_i such that $q_i \xleftarrow[\mathcal{A}]{*} t_i \to_{\mathcal{R}} t'_i \to_{\mathcal{A}}^* q'_i$. Because all q_j are reachable, we can fix terms t_j with $t_j \to_{\mathcal{A}}^* q_j$ for $j \neq i$. The state q is productive, so there is a context C such that $C[q] \to_{\mathcal{A}}^* q_\bullet \in Q_f$. Let $t = f(t_1, \ldots, t_n)$ and $t' = f(t_1, \ldots, t'_i, \ldots, t_n)$. Then $C[t] \in \mathcal{L}(\mathcal{A})$ and $C[t] \to_{\mathcal{R}} C[t']$, hence $C[t'] \in \mathcal{L}(\mathcal{A})$ as well. Consequently, there are states $q', q'_\bullet$ such that

$$C[q] \xleftarrow[\mathcal{A}]{*} C[t] \xrightarrow[\mathcal{R}]{} C[t'] \xrightarrow[\mathcal{A}]{*} C[f(q_1, \ldots, q'_i, \ldots, q_n)] \xrightarrow[\mathcal{A}]{} C[q'] \xrightarrow[\mathcal{A}]{*} q'_\bullet \in Q_f$$

In particular, we have a transition $f(q_1, \ldots, q'_i, \ldots, q_n) \to q' \in \Delta$, and $q \gg q'$.

3. (state-compatibility) Assume that $l\sigma \to_{\mathcal{A}}^* q$ for a state substitution σ. All states of $\mathcal{A}$ are reachable, so there is a substitution $\tau : \mathcal{V} \to \mathcal{T}(\mathcal{F})$ with $\tau(x) \to_{\mathcal{A}}^* \sigma(x)$ for all $x \in \mathcal{V}$. Furthermore, q is productive, so that for some context C, $C[q] \to_{\mathcal{A}}^* q_\bullet \in Q_f$. We have $C[l\tau] \in \mathcal{L}(\mathcal{A})$ and $C[l\tau] \to_{\mathcal{R}} C[r\tau]$. Consequently, $C[r\tau] \in \mathcal{L}(\mathcal{A})$ and for some states $q', q'_\bullet$,

$$C[q] \underset{\mathcal{A}}{\overset{*}{\leftarrow}} C[l\sigma] \underset{\mathcal{A}}{\overset{*}{\leftarrow}} C[l\tau] \underset{\mathcal{R}}{\longrightarrow} C[r\tau] \underset{\mathcal{A}}{\overset{*}{\rightarrow}} C[q'] \underset{\mathcal{A}}{\overset{*}{\rightarrow}} q'_\bullet \in Q_f$$

In particular, $r\tau \to_{\mathcal{A}}^* q'$. Recall that $\mathcal{A}$ is deterministic. Hence we can decompose this rewrite sequence as follows: $r\tau \to_{\mathcal{A}}^* r\sigma \to_{\mathcal{A}}^* q'$. We conclude by noting that $q \gg q'$ by the definition of $\gg$. □

Corollary 14. *The problem $\mathcal{R}(L) \subseteq L$ is decidable for finite tree automata $\mathcal{A}$.*

Proof. W.l.o.g. we may assume that $\mathcal{A}$ is deterministic. Using Proposition 4 we may also assume that $\mathcal{A}$ is trim. By Theorems 11 and 13 the problem reduces to whether there is some relation $\gg$ such that $(\mathcal{A}, \gg)$ is both state-compatible with $\mathcal{R}$ and state-coherent. But since there are only finitely many relations $\gg$ we can just test state-compatibility and state-coherence for each $\gg$. □

Remark 15. As a consequence of Theorem 13, regular languages accepted by state-coherent automata that are state-compatible with a fixed TRS $\mathcal{R}$ are closed under intersection and union. This can also be shown directly by a product construction.

3.4. Deciding $\mathcal{R}(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A})$

In the remainder of this section we show that instead of testing all possible relations $\gg$, it suffices to construct a minimal one. We proceed as follows:

1. We assume that $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ is finite, trim and deterministic. Note that given a non-deterministic automaton, we can compute an equivalent deterministic one in exponential time. Once we have a deterministic automaton, we can compute an equivalent trim one in polynomial time.
2. In the following steps we will find the smallest relation $\gg$ that makes $(\mathcal{A}, \gg)$ both state-compatible with $\mathcal{R}$ and state-coherent, if such a relation exists. Initially set $\gg = \emptyset$.
3. Consider each rule $l \to r \in \mathcal{R}$, each state substitution σ, and each state $q \in Q$ such that $l\sigma \to_{\Delta}^* q$. If there is some state q' with $r\sigma \to_{\Delta}^* q'$ then add (q, q') to $\gg$. Otherwise, $\mathcal{L}(\mathcal{A})$ is not closed under rewriting by $\mathcal{R}$, and the procedure terminates.
   At this point it is ensured that (any extension of) $\gg$ is state-compatible with $\mathcal{R}$.
4. In order to ensure state-coherence, we repeat the following process until $\gg$ is not increased any further.
   Whenever $q \gg q'$ and $f(q_1, \ldots, q_i = q, \ldots, q_n) \to q_\bullet \in \Delta$, then we look for a transition with left-hand side $f(q_1, \ldots, q'_i = q', \ldots, q_n)$ in Δ. If no such transition exists, state-coherence fails, and the algorithm terminates. Otherwise, let $q'_\bullet \in Q$ be the corresponding right-hand side and add $(q_\bullet, q'_\bullet)$ to $\gg$.
   At this point $\gg$ is the smallest relation satisfying state-compatibility with $\mathcal{R}$ which additionally satisfies the second condition of state-coherence.
5. Assert for all $q \gg q'$ with final state q that also q' is final.

Step 3 identifies the applicable instances of the state-compatibility constraint. It consists of a polynomial number of NP queries. Steps 4 and 5 can be performed in polynomial time. The whole procedure is, therefore, in the Δ_2^P (or P^{NP}) complexity class for deterministic automata as input.

Remark 16. Using [3, Exercise 1.12.2], which shows that it is NP-hard to decide whether an instance of a term l is accepted by a tree automaton $\mathcal{A}$, we can show that deciding whether the language accepted by a deterministic automaton is closed under rewriting by a given TRS is co-NP-hard. To wit, given a term l, a tree automaton $\mathcal{A}$, a fresh unary function $\star$ and a fresh constant $\diamond$, then $\star(\mathcal{L}(\mathcal{A})) = \{\star(x) \mid x \in \mathcal{L}(\mathcal{A})\}$ is closed under rewriting by $\star(l) \to \diamond$ if and only if no instance of l is accepted by $\mathcal{A}$.

4. Related work

4.1. Quasi-deterministic automata

In this section we relate deterministic state-coherent, state-compatible automata to quasi-deterministic automata by Korp and Middeldorp [13]. Our interest in quasi-deterministic automata is two-fold. First, they represent the state of the art in tree automata completion for non-left-linear TRSs. Secondly, the existing implementation of tree automata completion in $\mathsf{T_TT_2}$ and CSI is based on quasi-deterministic automata, and our goal is to certify the resulting non-confluence and

termination proofs (cf. Sections 5 and 6). We show that given a compatible, quasi-deterministic automaton, we can extract a state-compatible, deterministic automaton accepting the same language, while the opposite direction fails in the presence of collapsing rules. In our tools' implementation, due to Korp, this restriction is overcome by adding ε-transitions to the quasi-deterministic automata, and we will conclude the section with a description of this extension, which was hitherto unpublished.

First we recall the definitions of compatibility and quasi-determinism.

Definition 17 *(Definition 18 of [13]).* Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ be a tree automaton. For a left-hand side $l \in \mathrm{lhs}(\Delta)$ of a transition, we denote the set $\{q \mid l \to q \in \Delta\}$ of possible right-hand sides by $Q(l)$. We call $\mathcal{A}$ *quasi-deterministic* if for every $l \in \mathrm{lhs}(\Delta)$ there exists a *designated state* $p \in Q(l)$ such that for all transitions $f(q_1, \ldots, q_n) \to q \in \Delta$ and $i \in \{1, \ldots, n\}$ with $q_i \in Q(l)$, the transition $f(q_1, \ldots, q_{i-1}, p, q_{i+1}, \ldots, q_n) \to q$ belongs to Δ. Moreover, we require that $p \in Q_f$ whenever $Q(l)$ contains a final state.

For each $l \in \mathrm{lhs}(\Delta)$ we fix a designated state p_l satisfying the constraints of Definition 17. We denote the set of designated states by Q_d and the set $\{l \to p_l \mid l \in \mathrm{lhs}(\Delta)\}$ by Δ_d. The notion of compatibility used for quasi-deterministic tree automata is refined slightly from the standard one, Definition 1.

Definition 18 *(Definition 23 of [13]).* Let $\mathcal{R}$ be a TRS and L a language. Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ be a quasi-deterministic tree automaton. We say that $\mathcal{A}$ is *compatible* with $\mathcal{R}$ and L if $L \subseteq \mathcal{L}(\mathcal{A})$ and for each rewrite rule $l \to r \in \mathcal{R}$ and state substitution $\sigma : \mathcal{V}\mathrm{ar}(l) \to Q_d$ such that $l\sigma \to^*_{\Delta_d} q$ it holds that $r\sigma \to^*_{\Delta} q$.

To bring the notion of compatibility closer to our work, we say that $\mathcal{A}$ is compatible with $\mathcal{R}$ if $\mathcal{A}$ is compatible with $\mathcal{R}$ and $\varnothing$, making the condition $L \subseteq \mathcal{L}(\mathcal{A})$ vacuous. Example 7 exhibits a quasi-deterministic automaton that is compatible with $\mathcal{R}$.

We will show that for each quasi-deterministic automaton that is compatible with a TRS $\mathcal{R}$, there is a deterministic, state-coherent automaton that is state-compatible with $\mathcal{R}$ and accepts the same language. To this end, we need the following key lemma, a slight generalization of [13, Lemma 20], which shows that a quasi-deterministic automaton $\mathcal{A}$ is almost deterministic: all but the last step in a reduction can be performed using the deterministic Δ_d transitions.

Lemma 19. *Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ be a quasi-deterministic automaton. If $t \to^+_{\Delta} q$ then $t \to^*_{\Delta_d} \cdot \to_{\Delta} q$ for all terms $t \in \mathcal{T}(\mathcal{F} \cup Q)$ and states $q \in Q$.*

Proof. The proof is identical to the proof of [13, Lemma 20], except when t_i in $t = f(t_1, \ldots, t_n)$ is a state. In that case, we let $p_{l_i} = q_i = t_i$. □

Theorem 20. *Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ be a quasi-deterministic tree automaton that is compatible with $\mathcal{R}$. Then $\mathcal{A}' = (\mathcal{F}, Q_d, Q_f \cap Q_d, \Delta_d)$ makes $(\mathcal{A}', \gg)$ state-coherent and state-compatible with $\mathcal{R}$, where $q \gg q'$ if $q = q'$ or, for some left-hand side $l \in \mathrm{lhs}(\Delta)$, $q \in Q(l)$ and $q' = p_l$. Furthermore, $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Proof. Note that $\to_{\mathcal{A}} = \to_{\Delta}$ and $\to_{\mathcal{A}'} = \to_{\Delta_d}$.

1. (state-coherence) Assume that q is final in $\mathcal{A}'$, and $q \gg q'$. If $q = q'$ then q' is final, too. Otherwise, there is a left-hand side l such that $q \in Q(l)$ and $q' = p_l$ is the designated state of l. Since $Q(l)$ contains a final state (namely, q), q' must be final as well by Definition 17.
2. (state-coherence) Let $l = f(q_1, ..., q_i, ..., q_n)$ and $l' = f(q_1, ..., q'_i, ..., q_n)$, where $q_i \gg q'_i$. Furthermore, let $l \to q \in \Delta_d$. If $q_i = q'_i$ then $l' \to q \in \Delta_d$ and $q \gg q$. Otherwise, there is a left-hand side $l^\bullet$ such that $q_i \in Q(l^\bullet)$ and $q'_i = p_{l^\bullet}$ is the designated state of $l^\bullet$. By Definition 17, there is a transition $l' \to q$ in Δ. Thus, l' is a left-hand side and $q \in Q(l')$. Furthermore, $l' \to p_{l'} \in \Delta_d$, and $q \gg p_{l'}$ follows.
3. (state-compatibility) Let σ be a state substitution and $l\sigma \to^*_{\Delta_d} q$. By compatibility, we have $r\sigma \to^*_{\Delta} q$. If r is a variable, we are done, noting that $q \gg q$. Otherwise, using Lemma 19, there is a left-hand side $l' \in \mathrm{lhs}(\mathcal{A})$ such that $r\sigma \to^*_{\Delta_d} l' \to_{\Delta} q$. Consequently, $r\sigma \to^*_{\Delta_d} \cdot \to_{\Delta_d} p_{l'}$, and since $q \in Q(l')$, we have $q \gg p_{l'}$.
4. (accepted language) $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$ is obvious. To show $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$, assume that $t \in \mathcal{L}(\mathcal{A})$, i.e., $t \to^*_{\Delta} q \in Q_f$. By Lemma 19, there is a left-hand side $l \in \mathrm{lhs}(\mathcal{A})$ such that $t \to^*_{\Delta_d} l \to_{\Delta} q$. As in the previous item we conclude that $t \to^*_{\Delta_d} p_l$, and $q \gg p_l$. The state p_l is final by state-coherence, so $t \in \mathcal{L}(\mathcal{A}')$ follows. □

In the opposite direction, we have a positive result for non-collapsing TRSs.

Theorem 21. *Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ be a deterministic automaton and the relation $\gg \subseteq Q \times Q$ be such that $(\mathcal{A}, \gg)$ is state-coherent and state-compatible with $\mathcal{R}$. Furthermore, assume that $\mathcal{R}$ contains no collapsing rules. Then the automaton $\mathcal{A}' =$*

$(\mathcal{F}, Q, Q_f, \Delta')$ with $\Delta' = \{l \to q' \mid l \to q \in \Delta, q \gg^{=} q'\}$ is a quasi-deterministic automaton with designated states $p_l = q$ for $l \to q \in \Delta$, such that $\mathcal{A}'$ is compatible with $\mathcal{R}$ and accepts the same language as $\mathcal{A}$.

Proof. Verifying that the construction results in a quasi-deterministic automaton that is compatible with $\mathcal{R}$ is straightforward. Note that applying Theorem 20 to $\mathcal{A}'$ results in some $(\mathcal{A}'', \gg'')$ with $\mathcal{L}(\mathcal{A}'') = \mathcal{L}(\mathcal{A}')$, where $\mathcal{A}''$ is $\mathcal{A}$ with states restricted to Q'_d, the right-hand sides of Δ'. This restriction preserves the accepted language. Therefore, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. □

If $\mathcal{R}$ contains collapsing rules, quasi-deterministic, compatible automata may be weaker than state-coherent, state-compatible ones, as the following example demonstrates.

Example 22. Let $\mathcal{R} = \{\mathsf{f}(x, x) \to x\}$. The automaton $\mathcal{A}'$ over $\{\mathsf{f}, \mathsf{a}\}$ with states $1, 2$, both final, and transitions

$$\mathsf{a} \to 1 \qquad \mathsf{f}(1, 1) \to 2$$

accepts $L = \{\mathsf{f}(\mathsf{a}, \mathsf{a}), \mathsf{a}\}$. Furthermore, $(\mathcal{A}', \gg)$ is state-coherent and state-compatible with $\mathcal{R}$ if we let $2 \gg 1$.

Now assume that $\mathcal{A} = (\{\mathsf{f}, \mathsf{a}\}, Q, Q_f, \Delta)$ is a quasi-deterministic automaton and compatible with $\mathcal{R}$, and that $\mathsf{f}(\mathsf{a}, \mathsf{a}) \in \mathcal{L}(\mathcal{A})$. We will show that $\mathcal{A}$ accepts all terms over $\{\mathsf{f}, \mathsf{a}\}$. Note that since $\mathsf{f}(\mathsf{a}, \mathsf{a})$ is accepted, a must be a left-hand side of $\mathcal{A}$. Let q be the designated state of a. By Lemma 19, we have a run $\mathsf{f}(\mathsf{a}, \mathsf{a}) \to^*_{\Delta_d} \mathsf{f}(q, q) \to_\Delta q' \in Q_f$. Let $q^\bullet$ be the designated state of the left-hand side $\mathsf{f}(q, q)$. By quasi-determinism, $q^\bullet$ is a final state. Compatibility requires that $\mathsf{f}(q, q) \to_{\Delta_d} q^\bullet \;{}^*_\Delta\!\leftarrow q$, i.e., $q^\bullet = q$. So we have a final state q and two transitions $\mathsf{a} \to q$, $\mathsf{f}(q, q) \to q$, and $\mathcal{A}$ accepts all of $\mathcal{T}(\{\mathsf{f}, \mathsf{a}\})$.

Remark 23. In his thesis [12], Korp generalizes Definition 17 (cf. [12, Definition 3.10]) by incorporating an auxiliary relation $\succeq_{\phi_\mathcal{A}}$ that may be viewed as a precursor to our relation $\gg$. The modified definition permits smaller automata, which benefits implementations, but is more complicated than Definition 17. The modification also does not add expressive power. Indeed if $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ satisfies [12, Definition 3.10] using $\succeq_{\phi_\mathcal{A}}$, then taking $\Delta' = \{l \to q \mid l \in \mathrm{lhs}(\Delta), \phi_\mathcal{A}(l) \succeq q\}$, the automaton $\mathcal{A}' = (\mathcal{F}, Q, Q_f, \Delta')$ satisfies Definition 17, noting that $\phi_\mathcal{A}(l)$ is just another notation for the designated state p_l of l. Furthermore, $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.

The actual implementation of quasi-deterministic automata in TTT2 and CSI by Korp is based on an extension with ε-transitions, which are transitions from states to states.

Definition 24. Let $\mathcal{A}_\varepsilon = (\mathcal{F}, Q, Q_f, \Delta, \Delta_\varepsilon)$ be a tree automaton with ε-transitions $\Delta_\varepsilon \subseteq Q \times Q$. Let $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ be the corresponding tree automaton with the ε-transitions removed. Then $\mathcal{A}_\varepsilon$ is *extended quasi-deterministic* if $\mathcal{A}$ is quasi-deterministic and $l \to q' \in \Delta$ whenever $l \to q \in \Delta$ and $q \to q' \in \Delta_\varepsilon$.

We let $\to_{\mathcal{A}_\varepsilon} = \to_{\Delta \cup \Delta_\varepsilon}$. The following lemma is key to justifying the extension.

Lemma 25. *Let $\mathcal{A}_\varepsilon = (\mathcal{F}, Q, Q_f, \Delta, \Delta_\varepsilon)$ be an extended quasi-deterministic automaton with ε-transitions. Then for terms $t \in \mathcal{T}(\mathcal{F})$, we have $t \to^*_{\mathcal{A}_\varepsilon} q$ if and only if $t \to^*_\mathcal{A} q$.*

Proof. By structural induction on t. □

As a consequence, the language $\mathcal{L}(\mathcal{A}_\varepsilon) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \to^*_{\mathcal{A}_\varepsilon} q, q \in Q_f\}$ accepted by $\mathcal{A}_\varepsilon$ satisfies $\mathcal{L}(\mathcal{A}_\varepsilon) = \mathcal{L}(\mathcal{A})$. Furthermore, Lemmas 20 and 21 and Theorem 24 of [13] remain true for extended quasi-compatible automata, and therefore $\mathcal{L}(\mathcal{A}_\varepsilon)$ is closed under rewriting by $\mathcal{R}$ if $\mathcal{A}_\varepsilon$ is extended quasi-deterministic and compatible with $\mathcal{R}$. In order to obtain a deterministic, state-coherent, state-compatible automaton, we apply Theorem 20, and then extend $\gg$ as necessary to ensure compatibility with collapsing rules from $\mathcal{R}$. This is easy in our implementation, because ε-transitions correspond to state instances of collapsing rules. More precisely, $q' \to q$ is added only if $l \to x \in \mathcal{R}$, $l\sigma \to^*_{\mathcal{A}_\varepsilon} q$ and $\sigma(x) = q'$. A general approach is to observe that $(\mathcal{F}, Q, Q_f, \Delta_d)$ accepts the same language as $\mathcal{A}_\varepsilon$, and then compute $\gg$ by the algorithm from Section 3.4.

4.2. Partial quasi-models

In [4], Endrullis et al. introduce partial quasi-models for establishing *local termination*, i.e., termination on a restricted set of terms. To this end they consider *partial* $\mathcal{F}$-algebras, which consist of a carrier A and a partial function $[f] : A^n \to A$ for each $f \in \mathcal{F}$ of arity n. Let $[t, \alpha]$ be the (partial) evaluation map $\mathcal{T}(\mathcal{F}, \mathcal{V}) \times A^\mathcal{V} \to A'$.

Definition 26 *([4, Definition 7.2]).* A *partial quasi-model of a TRS $\mathcal{R}$* is a partial $\mathcal{F}$-algebra with carrier A equipped with a partial order $\geq$ such that

1. for all $\ell \to r \in \mathcal{R}$ and $\alpha : \mathcal{V} \to A$, if $[\ell, \alpha]$ is defined, $[\ell, \alpha] \geq [r, \alpha]$; and
2. $[f]$ is closed and monotone with respect to $\geq$ for all $f \in \mathcal{F}$.

Partial quasi-models over finite carriers A are almost the same as deterministic, state-compatible, state-coherent automata.[1] We briefly explain the connection between the two concepts.

Given a quasi-model for $\mathcal{R}$ with finite carrier A, we construct an equivalent state-coherent, state-compatible (with $\mathcal{R}$) automaton as follows. We take $Q = A$, $\gg = \geq$ and

$$\Delta = \{f(q_1, \ldots, q_n) \to [f](q_1, \ldots, q_n) \mid [f](q_1, \ldots, q_n) \text{ is defined}\}$$

It is straight-forward to prove that for ground terms t, $t \to_\Delta q$ if and only if $[t, \alpha]$ is defined and $[t, \alpha] = q$. Furthermore, the first condition of Definition 26 is equivalent to state-compatibility, while the second condition amounts to state-coherence, disregarding the restrictions on final states. This construction is invertible; given a deterministic automaton $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ and a partial order $\gg$ such that $(\mathcal{A}, \gg)$ is state-coherent and state-compatible with $\mathcal{R}$, we take $A = Q$, $\geq = \gg$ and let $[f](q_1, \ldots, q_n) = q$ whenever $f(q_1, \ldots, q_n) \to q \in \Delta$.

There are a few differences between deterministic state-coherent, state-compatible automata and quasi-models with finite carriers. First of all, tree automata come with a set of final states that is different from the set of states; this, crucially, allows them to accept languages that are not closed under the subterm relation. (This is not a problem in [4], where the languages of interest are terminating terms. These languages can be closed under subterms without losing the termination property.)

A second difference is that we do not require $\gg$ to be a partial order. However, this difference is not essential: First of all, by Remark 10, we may assume $\gg$ to be reflexive and transitive without loss of generality. Furthermore, if $q \gg q' \gg q$, then the state-coherence conditions force q and q' to be equivalent states. By mapping each state to its equivalence class with respect to $\equiv = \gg \cap \ll$, we obtain a new state-coherent, state-compatible (with $\mathcal{R}$) automaton equipped with the partial order $\gg/\equiv$.

Finally, note that we allow tree automata to be non-deterministic, and non-determinism does not map naturally to partial quasi-models (without effectively performing a powerset construction).

5. Confluence

Tree automata have an obvious application for disproving (local) confluence, as pointed out in [19]. Given some peak $s \,{}^*_{\mathcal{R}}\!\leftarrow \cdot \to^*_{\mathcal{R}} t$ (or $s \,{}_{\mathcal{R}}\!\leftarrow \cdot \to_{\mathcal{R}} t$ for local confluence), one has to prove that s and t are not joinable. To this end, it suffices to find suitable tree automata over-approximating descendants of s and t, respectively.

Observation 27. *Let $\mathcal{A}_s$ and $\mathcal{A}_t$ be tree automata. If $s \in \mathcal{L}(\mathcal{A}_s)$, $t \in \mathcal{L}(\mathcal{A}_t)$, $\mathcal{L}(\mathcal{A}_s) \cap \mathcal{L}(\mathcal{A}_t) = \emptyset$, and both automata are closed under rewriting with $\mathcal{R}$, then s and t are not joinable w.r.t. $\mathcal{R}$.*

Given the peak and both automata for a concrete TRS $\mathcal{R}$, by the decision procedure for closure under rewriting it is easy to check the conditions of Observation 27.

Notice that due to the precision of our criterion, we also can strengthen the power of confluence tools which are based on Observation 27 as demonstrated in the upcoming example.

Example 28. Consider the TRS $\mathcal{R}$ consisting of the following rules.

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

Disproving local confluence is equivalent to finding a non-joinable critical pair. Note that $\mathcal{R}$ contains only one non-joinable critical pair, arising from the peak $\mathsf{f}(\mathsf{a}, \mathsf{b}) \,{}_{\mathcal{R}}\!\leftarrow \mathsf{c} \to_{\mathcal{R}} \mathsf{f}(\mathsf{a}, \mathsf{a})$.

Proving non-joinability of $\mathsf{f}(\mathsf{a}, \mathsf{b})$ and $\mathsf{f}(\mathsf{a}, \mathsf{a})$ is possible via Observation 27 and the following two automata: the first automaton has one final state 3, and consists of four transitions, and accepts the language $\{\mathsf{f}(\mathsf{a}, \mathsf{b}), \mathsf{f}(\mathsf{b}, \mathsf{a})\}$.

$$\mathsf{a} \to 1 \qquad \mathsf{b} \to 2 \qquad \mathsf{f}(1, 2) \to 3 \qquad \mathsf{f}(2, 1) \to 3$$

The second automaton has three final states 1, 2, and 3, and also contains four transitions. Its language is $\{\mathsf{f}(\mathsf{a}, \mathsf{a}), \mathsf{f}(\mathsf{b}, \mathsf{b}), \mathsf{a}, \mathsf{b}\}$.

$$\mathsf{a} \to 1 \qquad \mathsf{b} \to 2 \qquad \mathsf{f}(1, 1) \to 3 \qquad \mathsf{f}(2, 2) \to 3$$

By Observation 27, $\mathcal{R}$ is not confluent. In the following we will argue that it is impossible to show non-confluence this way when using quasi-deterministic automata and compatibility for closure under rewriting.

Assume there is some quasi-deterministic automaton $\mathcal{A}$ with transitions Δ which accepts $\mathsf{f}(\mathsf{a}, \mathsf{a})$ and is compatible with $\mathcal{R}$. In the same way as in Example 22 one obtains transitions $\mathsf{a} \to q$ and $\mathsf{f}(q, q) \to q$, where q is a final state and the designated state of both a and $\mathsf{f}(q, q)$. Since $\mathcal{A}$ is closed under rewriting it must also contain $\mathsf{f}(\mathsf{b}, \mathsf{b})$ and thus, by the same

[1] This connection was pointed to the authors by J. Endrullis in personal communication.

reasoning there are transitions $\mathsf{b} \to p$ and $\mathsf{f}(p,p) \to p$ for some final state p which is also the designated state of b and $\mathsf{f}(p,p)$.

Since $\mathcal{A}$ is compatible and $\mathsf{f}(\mathsf{a},\mathsf{a}) \to^*_{\Delta_d} q$ we deduce $\mathsf{f}(\mathsf{b},\mathsf{b}) \to^*_{\Delta} q$. By Lemma 19 we further conclude $\mathsf{f}(\mathsf{b},\mathsf{b}) \to^*_{\Delta_d} \mathsf{f}(p,p) \to_{\Delta} q$. Hence, both p and q are contained in $Q(\mathsf{f}(p,p))$ where p is the designated state of $\mathsf{f}(p,p)$. Thus, by the definition of quasi-compatible automata, we can exchange q by p in any left-hand side of a transition. So, from $\mathsf{f}(q,q) \to q \in \Delta$ we know that also $\mathsf{f}(q,p) \to q \in \Delta$. Thus we obtain the derivation $\mathsf{f}(\mathsf{a},\mathsf{b}) \to^*_{\Delta} \mathsf{f}(q,p) \to_{\Delta} q$ which shows $\mathsf{f}(\mathsf{a},\mathsf{b}) \in \mathcal{L}(\mathcal{A})$. Therefore, no automaton which accepts $\mathsf{f}(\mathsf{a},\mathsf{b})$ is disjoint from $\mathcal{A}$.

6. Match-bounds

In this section we show how tree automata can be used to prove termination via match-bounds [9]. To this end, we first recapitulate the basic concepts and important results about match-bounds. Note that match-bounds require special treatment for non-left-linear TRSs (see Example 30). In [13], raise-consistent automata are used to ensure correctness. We present an adaptation of raise-consistency to our setting, leading to the new notion of state-raise-consistency. Finally, we explain how we treat quasi-compatibility, a weaker condition than compatibility that has been introduced specifically for match-bounds.

6.1. A short introduction to match-bounds

Match-bounds is a termination technique which is based on the following idea.

1. One considers an enriched signature where each original symbol of $\mathcal{F}$ is labeled by some natural number to yield a symbol of $\mathcal{F}' = \mathcal{F} \times \mathbb{N}$.
2. The rewrite rules of $\mathcal{R}$ over $\mathcal{F}$ are enriched by labels in a way that each rewrite step corresponds to an increase of labels.[2] The result is an enriched TRS $\mathcal{R}'$ over $\mathcal{F}'$. Possible enrichments are *match* and *roof* where the details are not relevant for this paper.
3. One tries to show boundedness, i.e., whenever one picks some initial term where all labels in t are 0, then there must be some bound b so that the labels never exceed b when rewriting with $\mathcal{R}'$.
4. If boundedness is ensured, then termination follows as any infinite derivation would lead to an infinite increase in the labels by 2., which is impossible by 3.

Termination via match-bounds can easily be treated as a tree automata problem: the set of terms where every symbol is labeled by 0, $\mathsf{lift}_0(\mathcal{F})$, is accepted by a tree automaton, and moreover, tree automata completion of $\mathsf{lift}_0(\mathcal{F})$ under $\to_{\mathcal{R}'}$, if successful, yields a suitable bound b for Step 3, namely the largest label in the transitions of the resulting automaton $\mathcal{A}$.

In short, match-bounds can be summarized as follows.

Theorem 29. *If $\mathcal{R}'$ is a valid enrichment of $\mathcal{R}$, $\mathcal{R}$ is left-linear, $\mathcal{A}$ is a finite tree automaton, $\mathsf{lift}_0(\mathcal{F}) \subseteq \mathcal{L}(\mathcal{A})$, and $\mathcal{A}$ is closed under $\to_{\mathcal{R}'}$, then $\mathcal{R}$ is terminating.*

It is well known that the restriction to left-linearity is essential as otherwise not every rewrite step of $\mathcal{R}$ can be simulated by a corresponding step in $\mathcal{R}'$.

Example 30. For $\mathcal{R} = \{\mathsf{f}(x,x) \to \mathsf{f}(\mathsf{a},x)\}$ and both possible enrichments we obtain $\mathcal{R}' = \{\mathsf{f}_i(x,x) \to \mathsf{f}_{i+1}(\mathsf{a}_{i+1},x) \mid i \in \mathbb{N}\}$. Now the derivation

$$\mathsf{f}(\mathsf{a},\mathsf{a}) \underset{\mathcal{R}}{\longrightarrow} \mathsf{f}(\mathsf{a},\mathsf{a}) \underset{\mathcal{R}}{\longrightarrow} \mathsf{f}(\mathsf{a},\mathsf{a}) \underset{\mathcal{R}}{\longrightarrow} \dots$$

cannot be simulated in the enriched system since after one step

$$\mathsf{f}_0(\mathsf{a}_0,\mathsf{a}_0) \underset{\mathcal{R}'}{\longrightarrow} \mathsf{f}_1(\mathsf{a}_1,\mathsf{a}_0)$$

there is a mismatch of the labels which cannot be repaired by $\mathcal{R}'$, and thus, the evaluation gets stuck.

To overcome this problem, we follow [13] and consider a special rewrite relation which allows to adjust labels for matching non-left-linear rules. In the following definition, base is the function which removes all labels of a term, and $s{\uparrow}t$ takes two terms with $\mathsf{base}(s) = \mathsf{base}(t)$ as input and performs a component-wise maximum on all labels. For example, $\mathsf{f}_1(\mathsf{a}_1,\mathsf{a}_3){\uparrow}\mathsf{f}_2(\mathsf{a}_0,\mathsf{a}_3) = \mathsf{f}_2(\mathsf{a}_1,\mathsf{a}_3)$. Moreover, for non-empty sets $S = \{s_1,\dots,s_n\}$ we define $\mathsf{base}(S) = \{\mathsf{base}(s_1),\dots,\mathsf{base}(s_n)\}$ and ${\uparrow}S = s_1{\uparrow}\dots{\uparrow}s_n$.

[2] There is an exception concerning collapsing rules as these do not increase the labels. This is explained in detail in [9].

Definition 31. Let $\mathcal{R}'$ be some TRS over an enriched signature $\mathcal{F}'$. We define the relation $\overset{\geqslant}{\to}_{\mathcal{R}'}$ as $s \overset{\geqslant}{\to}_{\mathcal{R}'} t$ if there is some rule $l \to r \in \mathcal{R}'$, $l = D\langle x_1, \ldots, x_n\rangle$ with all variables displayed, $s = C[D\langle s_1, \ldots, s_n\rangle]$, $S_i = \{s_j \mid 1 \leq j \leq n, x_j = x_i\}$ and $|\mathsf{base}(S_i)| = 1$ for each $1 \leq i \leq n$, $\tau = \{x_1/\uparrow S_1, \ldots, x_n/\uparrow S_n\}$, and $t = C[r\tau]$.

Note that τ in the previous definition is well-defined since whenever $x_i = x_j$ then $S_i = S_j$. Moreover, if $\mathcal{R}'$ is left-linear, then $\overset{\geqslant}{\to}_{\mathcal{R}'}$ is identical to $\to_{\mathcal{R}'}$. The purpose of $\overset{\geqslant}{\to}_{\mathcal{R}'}$ is to simulate rewriting steps using non-left-linear rules on enriched terms, even when matching fails due to mismatches in the labels as in Example 30.

Example 32. Recall Example 30, where $\mathsf{f}_1(\mathsf{a}_1, \mathsf{a}_0)$ could not be rewritten by the rule $\mathsf{f}_1(x, x) \to \mathsf{f}_2(\mathsf{a}_2, x)$ because x would have to equal both a_1 and a_0. With Definition 31 we have $\mathsf{f}_1(\mathsf{a}_1, \mathsf{a}_0) \overset{\geqslant}{\to}_{\mathcal{R}'} \mathsf{f}_2(\mathsf{a}_2, \mathsf{a}_1)$ using the rule $\mathsf{f}_1(x, x) \to \mathsf{f}_2(\mathsf{a}_2, x)$ and the substitution given by $\tau(x) = \mathsf{a}_1 \uparrow \mathsf{a}_0 = \mathsf{a}_1$.

Using this new relation it is possible to generalize Theorem 29 to arbitrary, possibly non-left-linear TRSs.

Theorem 33. *If $\mathcal{R}'$ is some valid enrichment of $\mathcal{R}$, $\mathcal{A}$ is some tree automaton, $\mathsf{lift}_0(\mathcal{F}) \subseteq \mathcal{L}(\mathcal{A})$, and $\mathcal{A}$ is closed under $\overset{\geqslant}{\to}_{\mathcal{R}'}$, then $\mathcal{R}$ is terminating.*

6.2. Adapting raise-consistency

The main difficulty when handling non-left-linear TRSs stems from the changed closure property where the standard rewrite relation $\to_{\mathcal{R}'}$ has been replaced by $\overset{\geqslant}{\to}_{\mathcal{R}'}$.

To handle this problem, the notion of raise-consistency was introduced in [13]. The basic idea is to ensure that whenever an automaton accepts terms s_1, s_2 with $\mathsf{base}(s_1) = \mathsf{base}(s_2)$, it also accepts $s_1 \uparrow s_2$ in a related state, cf. Lemma 35, thereby allowing to perform a step $s \overset{\geqslant}{\to}_{\mathcal{R}'} t$ step by first replacing s by another term s' accepted by the automaton (namely $s' = C[l\tau]$ in terms of Definition 31), followed by a plain rewrite step using a rule from $\mathcal{R}'$.

In the following we first adapt the notion of raise-consistency to our setting leading to the notion of state-raise-consistency, and prove that state-raise-consistency in combination with state-compatibility and state-coherence ensures closure under $\overset{\geqslant}{\to}_{\mathcal{R}'}$. Furthermore, we show that raise-consistent quasi-deterministic automata can easily be turned into state-raise-consistent deterministic automata.

Definition 34. $(\mathcal{A}, \gg)$ is *state-raise-consistent* if $q \gg q'$ for any transitions $f_i(q_1, \ldots, q_n) \to q$ and $f_j(q_1, \ldots, q_n) \to q'$ of $\mathcal{A}$ with $i < j$.

Until Corollary 37 we assume a fixed deterministic automaton $\mathcal{A}$, a TRS $\mathcal{R}'$, and a relation $\gg$, where $(\mathcal{A}, \gg)$ is state-raise-consistent, state-coherent, and state-compatible w.r.t. $\mathcal{R}'$. Following the basic idea outlined above, we show that $\mathcal{L}(\mathcal{A})$ is closed under rewriting by $\overset{\geqslant}{\to}_{\mathcal{R}'}$. Lemma 35 deals with combining the transitions $s_i \to^*_{\mathcal{A}} q_i$ to accept s, where s is decomposed into $C[D\langle s_1, \ldots, s_n\rangle]$ for a step $s \overset{\geqslant}{\to}_{\mathcal{R}'} t$ according to Definition 31.

Lemma 35. *Let $S = \{s_1, \ldots, s_n\}$ with $|\mathsf{base}(S)| = 1$. If $s_i \to^*_{\mathcal{A}} q_i$ for all $1 \leq i \leq n$ then there is some q such that $\uparrow S \to^*_{\mathcal{A}} q$ and $q_i \gg^* q$ for all $1 \leq i \leq n$*

Proof. We only consider the case $n = 2$ here, which is then easily generalized to arbitrary n. So, let $\mathsf{base}(s_1) = \mathsf{base}(s_2)$, and $s_i \to^*_{\mathcal{A}} q_i$ for both $i = 1, 2$. We perform induction on s_1, so let $s_1 = f_a(t_1, \ldots, t_k) \to^*_{\mathcal{A}} f_a(p_1, \ldots, p_k) \to_{\mathcal{A}} q_1$ and $s_2 = f_b(t'_1, \ldots, t'_k) \to^*_{\mathcal{A}} f_b(p'_1, \ldots, p'_k) \to_{\mathcal{A}} q_2$ where $\mathsf{base}(t_i) = \mathsf{base}(t'_i)$ for all $1 \leq i \leq k$. Hence, by the induction hypothesis we obtain q'_i such that $t_i \uparrow t'_i \to^*_{\mathcal{A}} q'_i$, $p_i \gg^* q'_i$, and $p'_i \gg^* q'_i$ for each i. Thus, $s_1 \uparrow s_2 = f_{\max(a,b)}(t_1 \uparrow t'_1, \ldots, t_k \uparrow t'_k) \to^*_{\mathcal{A}} f_{\max(a,b)}(q'_1, \ldots, q'_k)$. Using $f_a(p_1, \ldots, p_k) \to_{\mathcal{A}} q_1$, $p_i \gg^* q'_i$ and state-coherence, there is some $q^\bullet_1$ such that $f_a(q'_1, \ldots, q'_k) \to_{\mathcal{A}} q^\bullet_1$ and $q_1 \gg^* q^\bullet_1$. Similarly, we obtain $q^\bullet_2$ such that $f_b(q'_1, \ldots, q'_k) \to_{\mathcal{A}} q^\bullet_2$ and $q_2 \gg^* q^\bullet_2$.

It remains to prove existence of some $q^\bullet$ such that $f_{\max(a,b)}(q'_1, \ldots, q'_k) \to_{\mathcal{A}} q^\bullet$ and $q^\bullet_i \gg^* q^\bullet$ for both $i = 1, 2$: then we would be able to derive the desired result $s_1 \uparrow s_2 \to^*_{\mathcal{A}} f_{\max(a,b)}(q'_1, \ldots, q'_k) \to_{\mathcal{A}} q^\bullet$ and $q_i \gg^* q^\bullet_i \gg^* q^\bullet$ for both i. To show the existence of $q^\bullet$ we distinguish cases depending on how a and b compare.

If $a = b$, then $f_a(q'_1, \ldots, q'_k) \to_{\mathcal{A}} q^\bullet_1$, and $f_b(q'_1, \ldots, q'_k) \to_{\mathcal{A}} q^\bullet_2$ imply $q^\bullet_1 = q^\bullet_2$ by determinism of $\mathcal{A}$. Hence, we can let $q^\bullet := q^\bullet_1 = q^\bullet_2$ and are done. If $a < b$, then by state-raise-consistency we conclude $q^\bullet_1 \gg q^\bullet_2$ and choose $q^\bullet := q^\bullet_2$ to derive the desired result: $f_{\max(a,b)}(q'_1, \ldots, q'_k) = f_b(q'_1, \ldots, q'_k) \to_{\mathcal{A}} q^\bullet_2 = q^\bullet$ and $q^\bullet_i \gg^* q^\bullet_2 = q^\bullet$ for both i. The final case, $a > b$, is symmetric to $a < b$. □

We are no ready to deal with a full step $s \xrightarrow{\geqslant}_{\mathcal{R}'} t$.

Lemma 36. *If $s \to^*_{\mathcal{A}} q$ and $s \xrightarrow{\geqslant}_{\mathcal{R}'} t$, then there exists some q' with $t \to^*_{\mathcal{A}} q'$ and $q \gg^* q'$.*

Proof. Since $s \xrightarrow{\geqslant}_{\mathcal{R}'} t$ there are $l \to r \in \mathcal{R}'$ and C, D, s_i, S_i, and τ satisfying the conditions of Definition 31. Hence, we can decompose $s \to^*_{\mathcal{A}} q$ into $s = C[D\langle s_1, \ldots, s_n\rangle] \to^*_{\mathcal{A}} C[D\langle q_1, \ldots, q_n\rangle] \to^*_{\mathcal{A}} C[p] \to^*_{\mathcal{A}} q$, where $s_i \to^*_{\mathcal{A}} q_i$ for each $1 \leq i \leq n$ and $D\langle q_1, \ldots, q_n\rangle \to^*_{\mathcal{A}} p$. Since $|\mathsf{base}(S_i)| = 1$ we use Lemma 35 to obtain for each i some q'_i such that $\uparrow S_i \to^*_{\mathcal{A}} q'_i$ and $q_i \gg^* q'_i$. Moreover, since $S_i = S_j$ whenever $x_i = x_j$ we can ensure that $q'_i = q'_j$ whenever $x_i = x_j$. This allows us to define $\sigma = \{x_1/q'_1, \ldots, x_n/q'_n\}$ and we conclude both $D\langle q'_1, \ldots, q'_n\rangle = l\sigma$ and $\tau(x_i) \to^*_{\mathcal{A}} \sigma(x_i)$ for all $1 \leq i \leq n$.

From $D\langle q_1, \ldots, q_n\rangle \to^*_{\mathcal{A}} p$, $q_i \gg^* q'_i$ and state-coherence we obtain a p_0 satisfying $l\sigma = D\langle q'_1, \ldots, q'_n\rangle \to^*_{\mathcal{A}} p_0$ and $p \gg^* p_0$. From $l\sigma \to^*_{\mathcal{A}} p_0$ and state-compatibility we conclude that there is some p' such that $r\sigma \to^*_{\mathcal{A}} p'$ and $p_0 \gg p'$ and hence $p \gg^* p'$. In the same way, from $C[p] \to^*_{\mathcal{A}} q$, and $p \gg^* p'$ we obtain some q' such that $C[p'] \to^*_{\mathcal{A}} q'$ and $q \gg^* q'$.

This finally yields $t = C[r\tau] \to^*_{\mathcal{A}} C[r\sigma] \to^*_{\mathcal{A}} C[p'] \to^*_{\mathcal{A}} q'$ where $q \gg^* q'$. □

Corollary 37. *If $s \in \mathcal{L}(\mathcal{A})$ and $s \xrightarrow{\geqslant}_{\mathcal{R}'} t$, then $t \in \mathcal{L}(\mathcal{A})$.*

Therefore, we have the following criterion for termination of $\mathcal{R}$.

Corollary 38. *Let $\mathcal{R}$ be a TRS and $\mathcal{R}'$ a suitable enrichment for $\mathcal{R}$. If $\mathcal{A}$ is a finite deterministic tree automaton and $(\mathcal{A}, \gg)$ is state-raise-consistent, state-coherent, and state-compatible with $\mathcal{R}'$, and furthermore $\mathsf{lift}_0(\mathcal{F}) \subseteq \mathcal{L}(\mathcal{A})$ then $\mathcal{R}$ is terminating.*

Observe that Corollary 37 shows that state-coherence together with state-compatibility, state-raise-consistency, and determinism is a sufficient criterion for closure under rewriting with $\xrightarrow{\geqslant}_{\mathcal{R}'}$. However, it is not a necessary criterion, so there is no analogue of Theorem 13 where we replace $\to_{\mathcal{R}'}$ by $\xrightarrow{\geqslant}_{\mathcal{R}'}$ and add state-raise-consistency. This is demonstrated in the following example.

Example 39. Let $\mathcal{A}$ be the deterministic and trim automaton with transitions $\mathsf{a}_0 \to 0, \mathsf{a}_1 \to 1, \mathsf{f}_0(0) \to 2$ and final states $1, 2$. It accepts the language $L = \{\mathsf{a}_1, \mathsf{f}_0(\mathsf{a}_0)\}$. Let $\mathcal{R}' = \emptyset$. Then obviously $\mathcal{L}(\mathcal{A}) = L$ is closed under rewriting w.r.t. $\xrightarrow{\geqslant}_{\mathcal{R}'}$.

Now assume there is some relation $\gg$ such that $(\mathcal{A}, \gg)$ is both state-coherent and state-raise-consistent. From the latter and the transitions $\mathsf{a}_0 \to 0$ and $\mathsf{a}_1 \to 1$ we conclude $0 \gg 1$. In combination with the transition $\mathsf{f}_0(0) \to 2$ and state-coherence there must be some state q satisfying $\mathsf{f}_0(1) \to q$ and $2 \gg q$. However, since there is no transition with left-hand side $\mathsf{f}_0(1)$ we derived a contradiction, and thus there is no such relation $\gg$.

We present another example that shows that the match-bounds technique potentially suffers from this limitation.

Example 40. Consider the TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{g}(\mathsf{g}(x))) \to \mathsf{f}(\mathsf{g}(x))\}$, whose enrichment is $\mathcal{R}' = \{\mathsf{f}_a(\mathsf{g}_b(\mathsf{g}_c(x))) \to \mathsf{f}_d(\mathsf{g}_d(x)) \mid d = \min(a, b, c) + 1\}$. Let the automaton $\mathcal{A}$ have states $1, 2$, both final, and the following transitions:

$$\mathsf{f}_0(1) \to 1 \qquad \mathsf{g}_0(1) \to 1 \qquad \mathsf{c}_0 \to 1$$
$$\mathsf{f}_1(2) \to 1 \qquad \mathsf{g}_1(1) \to 2$$

With $\gg$ as equality, $\mathcal{A}$ is deterministic, state-coherent, and state-compatible with $\mathcal{R}'$, and therefore $\mathcal{R}$ is match-bounded.

However, there is no $(\mathcal{A}, \gg)$ that is finite, deterministic, state-coherent, state-compatible with $\mathcal{R}'$, and also state-raise-consistent, and accepts $\mathsf{f}_0(\mathsf{g}_0^k(\mathsf{c}_0))$ for all $k > 0$. To see why, let q_i be the state accepting $\mathsf{g}_0^i(\mathsf{c}_0)$. There is a state q with $\mathsf{f}_0(\mathsf{g}_0(\mathsf{g}_0(q_i))) \to^*_{\mathcal{A}} q$, and by state-compatibility with $\mathcal{R}'$, there are states q', q'' such that $\mathsf{f}_1(\mathsf{g}_1(q_i)) \to_{\mathcal{A}} \mathsf{f}_1(q') \to_{\mathcal{A}} q''$, where $q \gg q''$. In particular, there are transitions $\mathsf{g}_1(q_i) \to q'$, and $\mathsf{g}_0(q_i) \to q_{i+1}$, which implies $q_{i+1} \gg q'$ by state-raise-consistency. By state-coherence this implies that whenever $\mathcal{A}$ accepts $C[\mathsf{g}_0(q_i)]$, then it also accepts $C[\mathsf{g}_1(q_i)]$. We have $\mathsf{f}_1(\mathsf{g}_1(\mathsf{g}_0^k(\mathsf{c}_0)))$ in $\mathcal{L}(\mathcal{A})$ by closure under rewriting, and by induction on j we can now show that $\mathsf{f}_1(\mathsf{g}_1^{j+1}(\mathsf{g}_0^{k-j}(\mathsf{c}_0))) \in \mathcal{L}(\mathcal{A})$. In particular, $\mathsf{f}_1(\mathsf{g}_1^{k+1}(\mathsf{c}_0))$ is accepted by $\mathcal{A}$ as well.

By iterating this construction (increasing all labels of f and g by 1 in each iteration), we can show that $\mathcal{A}$ accepts $\mathsf{f}_a(\mathsf{g}_a^k(\mathsf{c}_0))$ for all $k > 0$ and $a \in \mathbb{N}$, which requires infinitely many transitions (at least one for each of the symbols f_a and g_a), contradicting finiteness of $\mathcal{A}$.

Despite incompleteness, state-raise-consistency still subsumes the criterion of raise-consistency for quasi-deterministic automata.

Definition 41 *([13, Definition 27]).* Let $\mathcal{A} = (\mathcal{F}', Q, Q_f, \Delta)$ be a tree automaton. We say that $\mathcal{A}$ is *raise-consistent* if for every pair of transitions $f_i(q_1, \ldots, q_n) \to q$ and $f_j(q_1, \ldots, q_n) \to q'$ in Δ with $i < j$, the transition $f_j(q_1, \ldots, q_n) \to q$ belongs to Δ.

Theorem 42. *Let $\mathcal{A}$, $\mathcal{R}$, $\mathcal{A}'$, and $\gg$ be as in Theorem 20. If $\mathcal{A}$ is raise-consistent then $(\mathcal{A}', \gg)$ is state-raise-consistent.*

Proof. Let $f_i(q_1, \ldots, q_n) \to q \in \Delta_d$ and $f_j(q_1, \ldots, q_n) \to q' \in \Delta_d$ be rules of $\mathcal{A}'$ with $i < j$. Hence, both rules are also present in $\mathcal{A}$ and by raise-consistency we conclude that $f_j(q_1, \ldots, q_n) \to q \in \Delta$. By definition of Δ_d we know that $q' = Q(f_j(q_1, \ldots, q_n))$ is the designated state and thus, $q \gg q'$ by the definition of $\gg$. □

Similarly to Example 22 we further prove that the inclusion is strict by providing a rewrite system where match-boundedness cannot be proven using quasi-deterministic automata.

Example 43. Let $\mathcal{R} = \{\mathsf{f}(x, x) \to x, \mathsf{f}(\mathsf{f}(x, y), z) \to \mathsf{f}(\mathsf{a}, \mathsf{a})\}$ over signature $\mathcal{F} = \{\mathsf{a}, \mathsf{f}\}$. Then the enriched system is $\mathcal{R}' = \{\mathsf{f}_i(x, x) \to x, \mathsf{f}_i(\mathsf{f}_j(x, y), z) \to \mathsf{f}_k(\mathsf{a}_k, \mathsf{a}_k) \mid i, j, k \in \mathbb{N}, k = 1 + \min(i, j)\}$. In particular, $\mathcal{R}'$ contains the rules $\mathsf{f}_i(x, x) \to x$ and $\mathsf{f}_i(\mathsf{f}_i(x, y), z) \to \mathsf{f}_{i+1}(\mathsf{a}_{i+1}, \mathsf{a}_{i+1})$ for all $i \in \mathbb{N}$.

The deterministic automaton $\mathcal{A}$ over $\{\mathsf{f}, \mathsf{a}\}$ with states $1, 2$, both final, and transitions

$$\mathsf{a}_0 \to 2 \qquad \mathsf{f}_0(p, q) \to 1 \quad \text{for all } p, q \in \{1, 2\}$$
$$\mathsf{a}_1 \to 2 \qquad \mathsf{f}_1(2, 2) \to 1$$

accepts all terms in $\mathsf{lift}_0(\mathcal{F})$ and is closed under rewriting with $\xrightarrow{\geqslant}_{\mathcal{R}'}$, as $(\mathcal{A}, \gg)$ is state-coherent and state-compatible with $\mathcal{R}'$ and state-raise-consistent if we let $1 \gg 1$ and $1 \gg 2$. Hence, by Theorem 33 and Corollary 37 termination of $\mathcal{R}$ is proved.

We further prove that a similar proof is impossible if we use compatible, quasi-deterministic automata. To this end, assume that $\mathcal{A} = (\mathcal{F} \times \mathbb{N}, Q, Q_f, \Delta)$ is a finite, quasi-deterministic automaton which is compatible[3] with $\mathcal{R}$ and accepts all terms in $\mathsf{lift}_0(\mathcal{F})$. These conditions imply that $\mathcal{L}(\mathcal{A})$ is closed under rewriting with $\mathcal{R}'$. Obviously, $\mathsf{f}_0(\mathsf{f}_0(\mathsf{a}_0, \mathsf{a}_0), \mathsf{a}_0) \in \mathcal{L}(\mathcal{A})$ and since $\mathsf{f}_0(\mathsf{f}_0(\mathsf{a}_0, \mathsf{a}_0), \mathsf{a}_0) \to_{\mathcal{R}'} \mathsf{f}_1(\mathsf{a}_1, \mathsf{a}_1)$ and $\mathcal{L}(\mathcal{A})$ is closed under rewriting, we also have $\mathsf{f}_1(\mathsf{a}_1, \mathsf{a}_1) \in \mathcal{L}(\mathcal{A})$.

Since $\mathsf{f}_1(x, x) \to x \in \mathcal{R}'$, we can proceed in the same way as in Example 22 to show that $\mathcal{A}$ accepts all terms over $\{\mathsf{f}_1, \mathsf{a}_1\}$, i.e., $\mathsf{lift}_1(\mathcal{F})$.

Now, again we have a derivation $\mathsf{f}_1(\mathsf{f}_1(\mathsf{a}_1, \mathsf{a}_1), \mathsf{a}_1) \to_{\mathcal{R}'} \mathsf{f}_2(\mathsf{a}_2, \mathsf{a}_2)$ and by closure under rewriting we conclude $\mathsf{f}_2(\mathsf{a}_2, \mathsf{a}_2) \in \mathcal{L}(\mathcal{A})$ and afterwards derive $\mathsf{lift}_2(\mathcal{F}) \subseteq \mathcal{L}(\mathcal{A})$ as before. Iterating this reasoning yields $\bigcup_{i \in \mathbb{N}} \mathsf{lift}_i(\mathcal{F}) \subseteq \mathcal{L}(\mathcal{A})$. But this is impossible, since $\mathcal{A}$ is a finite automaton which can only have finitely many symbols in the transitions whereas $\bigcup_{i \in \mathbb{N}} \mathsf{lift}_i(\mathcal{F})$ contains infinitely many symbols.

6.3. Quasi-compatibility

In [13, Section 5] the improvement of quasi-compatibility is introduced, which relaxes the compatibility criterion and therefore allows to reduce the size of the automata. While it is possible to also integrate this refinement into state-compatibility, we omit the details here. The main reason is that every quasi-(state-)compatible automata can easily be turned into an automaton which is also (state-)compatible by just adding more transitions, cf. the remark between Definitions 32 and 33 in [13]. Thus, in the same way as we transform quasi-deterministic automata into deterministic automata within Section 4.1, we can also always transform quasi-compatible automata into compatible ones without losing power.

7. Formalization

We have formalized all results from Section 3 and significant parts of Sections 5 and 6 within IsaFoR, our *Isa*belle *F*ormalization of *R*ewriting, in combination with executable algorithms which check state-compatibility, state-coherence, and state-raise-consistency. These are used in CeTA [18], a certifier for several properties related to term rewriting.

On the analyzer side we have extended the termination tool $\mathsf{T_TT_2}$ [11] and the confluence tool CSI [19] to produce state-coherent, state-compatible automata. Since both tools use quasi-deterministic automata in their completion process, we apply the construction of Theorem 20 as a post-processing step, resulting in a state-coherent, state-compatible automaton. CeTA can then be used to certify this output. In contrast to the earlier version of CeTA corresponding to [5], CeTA now supports match-bounds for non-left-linear TRSs as well.

All tools and the formalization are available at http://cl-informatik.uibk.ac.at/research/software/ (CeTA + IsaFoR version 2.23, $\mathsf{T_TT_2}$ version 1.15, CSI version 0.5.1.)

[3] We do not even require raise-consistency of $\mathcal{A}$.

Table 1
Definitions and properties which are available in IsaFoR.

Definition or property	Content
Preliminaries	basic notions on term rewriting and tree automata
Algorithms	membership, intersection, emptiness of tree aut.
Definition 3	reachable and productive states
Proposition 4	algorithm to compute trim automaton
Definition 8	state-compatibility and state-coherence
Remark 10	state-coherence of reflexive transitive closure
Theorem 11	soundness of state-coherence and state-compat.
Theorem 13	completeness of state-coherence and state-compat.
Corollary 14	decidability of closure under rewriting
Section 3.4	decision procedure of closure under rewriting
Observation 27	disproving joinability via tree automata
Theorem 29	soundness of match-bounds, left-linear version
Definition 31	rewrite relation $\overset{\geqslant}{\to}_{\mathcal{R}'}$
Theorem 33	soundness of match-bounds, non-left-linear version
Definition 34	state-raise-consistency
Lemmas 35 and 36	simulating $\overset{\geqslant}{\to}_{\mathcal{R}'}$-steps
Corollaries 37 and 38	sufficient criterion for match-bounds

In the following, we illustrate in more detail which parts of the paper are available in IsaFoR. Furthermore we motivate our design choices in the formalization, and elaborate on problems w.r.t. executability of the algorithms. We also compare our formalization to related work. The majority of the formalization is located in the files `Tree_Automata.thy`, `Tree_Automata_Impl.thy`, and `Raise_Consistency.thy`.

7.1. Coverage

Table 1 lists all the definitions and properties from the paper that have been formalized in IsaFoR.

Note that it does neither contain explicit notions for compatibility nor for quasi-deterministic tree automata, and hence also the various counterexamples from the paper are not contained in IsaFoR. Still, compatibility is supported by CeTA as it is simulated by state-compatibility and state-coherence where $\gg$ is chosen as the identity relation. Then state-coherence is vacuously satisfied and state-compatibility coincides with compatibility. In the case of quasi-deterministic automata, the tools are required to apply the conversion (Theorems 20 and 42) into state-compatible and state-coherent deterministic automata in their output.

Concerning the formalization of the decision procedure for closure under rewriting, there is a restriction on the input: CeTA demands that the tree automaton is already deterministic. The major reason is that at the moment we only formalized a non-optimized version of the powerset-construction which turns each automaton into an equivalent deterministic one: it always converts an automaton with $|Q|$ states into an automaton with $|2^Q|$ states.

7.2. Design choices and deviations

The formalization of basic definitions on tree automata contains two major differences to this paper: it allows ε-transitions, and the set of *re*achable *s*tates $t \to_\Delta^* q$ is formalized directly as a function *ta_res* mapping terms to sets of states. Using a function instead of a relation has both positive and negative effects. For example, it eases proofs which are naturally performed by induction on terms, since in $f(t_1, \ldots, t_n)$ one does not have to reduce all arguments t_1 to t_n sequentially in a relation, but this is done in one step in *ta_res*. On the other hand, one cannot trace derivations $t \to_\Delta^* q$ explicitly as there is no notion of derivation. Hence, some obvious results have to be proven explicitly by induction, e.g., that removing transitions results in a smaller accepted language.

Note an interesting subtlety in the following definition of the language of a tree automaton, *ta_lang*, namely the demand of the function *adapt_vars*. The definition of a language is nearly straight-forward:

$$ta\_lang\ \mathcal{A} = \{adapt\_vars\ t \mid ground\ t \wedge ta\_res\ \mathcal{A}\ t \cap ta\_final\ \mathcal{A} \neq \emptyset\}$$

where the function *adapt_vars* does nothing except changing the type of variables of ground terms.

The necessity of *adapt_vars* is motivated as follows. In the formalization we encode mixed expressions like $\mathsf{f}(q_1, \mathsf{a}, q_2)$ consisting of states q_1, q_2 and function symbols f, a simply as terms, where states are just represented as variables. The advantage of this encoding is the reusability of IsaFoR's library on terms. However, with this encoding, Isabelle's type system enforces that the type of variables of t in the definition of *ta_lang* is exactly the type of states, even if t is a ground term. Hence, if one removed *adapt_vars* from the definition of *ta_lang*, the type system of Isabelle would enforce that the type of states in $\mathcal{A}$ must correspond to the type of variables in *ta_lang* $\mathcal{A}$. As a result, soundness theorems for the powerset-construction would not even type-check, since the type of states changes from α (Q) to $\alpha\ set$ (2^Q), and thus, the two languages would have terms with variables of these different types, while the comparison requires equal types. Thanks

to *adapt_vars*, the type of variables in the accepted language may be different from the type of states of the automata, resolving this issue. This comes at a price: one sometimes has to reason within proofs that *adapt_vars* can be treated like an identity.

Another difference between paper and formalization becomes visible in Corollary 14. The formalization only states that $\mathcal{L}(\mathcal{A})$ is closed under $\mathcal{R}$ if and only if for the determinized and trimmed automaton there exists a suitable relation $\gg$. Instead of formally proving decidability by an algorithm which enumerates all possible relations, we directly formalized the algorithm of Section 3.4 as a function to compute the least such relation.

7.3. Executability

Many of the algorithms have been defined in two steps: first they have been formalized on an abstract level, which eased the corresponding soundness proofs; and later on they have been refined to fully executable ones. In fact, for some algorithms we just relied on the automatic refinement provided in [16] which turns set operations into operations on trees. For other algorithms we performed manual data refinement. Although the latter approach is more tedious, it has the advantage for the user that we additionally integrated detailed error messages which are displayed if the certifier rejects a proof. For example, when using the decision procedure which is mainly implemented via automatic refinement, we only get one bit of information; in contrast, the algorithm for ensuring state-compatibility for a user-provided relation $\gg$ returns a detailed reason in case of a state-compatibility violation.

For some computationally expensive algorithms we performed additional manual refinement steps to increase the efficiency. For example, we group the transitions of an automaton by their root symbols and store these groups in ordered trees using Isabelle's collection framework [15]. Moreover, for each $f(q_1, \ldots, q_n) \to q$, we precompute the closure of q under ε-transitions. This speeds up the computation of *ta_res* while checking state-compatibility. In the end, we provide an executable algorithm which for given $\mathcal{A}$ and $\mathcal{R}$ checks whether $\mathcal{A}$ is closed under $\mathcal{R}$. Isabelle's code generator transforms this algorithm—and several others—into Haskell code, which in combination with a small hand-written file is the source code of our certifier CeTA.

In the following we provide more details on specific algorithms which are essential for this paper.

The decision procedure requires a trim automaton as input. Here, CeTA is able to compute a trim automaton on its own, since Proposition 4 has been proven constructively. To be more precise, it is first shown that restricting an automaton to the set of reachable or productive states does not change the languages,

$$ta\_lang\ (ta\_only\_reach\ \mathcal{A}) = ta\_lang\ \mathcal{A}$$

$$ta\_lang\ (ta\_only\_prod\ \mathcal{A}) = ta\_lang\ \mathcal{A}$$

where *ta_only_prod* $\mathcal{A}$ and *ta_only_reach* $\mathcal{A}$ are the automata that are obtained from $\mathcal{A}$ by only keeping productive and reachable states, respectively. Then one easily defines[4]

$$trim\_ta\ \mathcal{A} = ta\_only\_prod\ (ta\_only\_reach\ \mathcal{A})$$

which turns an automaton into an equivalent trim one. Whereas equivalence is trivial with the help of the previous two properties, showing that the result is trim requires a connection between the restriction on reachable and productive states: We formalized that if in $\mathcal{A}$ all states are reachable then also in *ta_only_prod* $\mathcal{A}$ all states are reachable.

However, to make *trim_ta* executable, we actually need executable versions of *ta_only_prod* and *ta_only_reach*. For the former, we proved that productivity can be expressed in terms of a reflexive transitive closure which is then executable. For the latter, we implemented a working list algorithm which iteratively removes reachable states from the automaton. Note that in both cases, we did not define new recursive functions, but just proved equalities which are treated as function definitions by Isabelle's code generator [10].

The formalization of the algorithm of Section 3.4 (*decide_coherent_compatible*) is based on an inductive predicate that constitutes an abstract description of the decision procedure. Its soundness and completeness manifests in the following theorem.

$$\begin{aligned} &ta\_det\ \mathcal{A} \Longrightarrow finite\ (ta\_states\ \mathcal{A}) \Longrightarrow \\ &\quad decide\_coherent\_compatible\ \mathcal{A}\ \mathcal{R} \longleftrightarrow \\ &\quad (\exists \gg .\ state\_compatible\ \mathcal{A} \gg \mathcal{R} \wedge state\_coherent\ \mathcal{A} \gg) \end{aligned}$$

The abstract algorithm is later on refined to a fully executable one via implementing a working list algorithm to create a suitable relation $\gg$. It starts with all pairs of states that are enforced by state-compatibility and then iteratively adds new pairs of states which are demanded according to state-coherence.

[4] This definition was simplified a lot in comparison to [5] where *trim_ta* was defined as a recursive algorithm.

Combining *trim_ta* with *decide_coherent_compatible* yields the decision procedure *closed_under_rewriting*, an executable algorithm with the desired soundness property:

$$\textit{ta\_det}\ \mathcal{A} \Longrightarrow \textit{finite}\ (\textit{ta\_states}\ \mathcal{A}) \Longrightarrow$$
$$\textit{closed\_under\_rewriting}\ \mathcal{A}\ \mathcal{R} \longleftrightarrow (\to_{\mathcal{R}}\ (\textit{ta\_lang}\ \mathcal{A}) \subseteq \textit{ta\_lang}\ \mathcal{A})$$

In addition to the decision procedure, we also provide Theorem 11 to demonstrate closure under rewriting when $\gg$ is supplied. The advantage of the latter is its improved runtime and its broader applicability: one does not have to iteratively construct the relation, and for left-linear TRSs, also non-deterministic automata with ε-transitions are supported. Here, for checking state-compatibility, we use a tree automaton matching algorithm that restricts the set of state substitutions σ that have to be considered for compatibility w.r.t. Definition 8.

7.4. Related formalization

Note that there already exists another Isabelle library on tree automata [14,15] by Peter Lammich. We briefly relate the two libraries. Lammich's formalization is more complete in terms of pure tree automata operations: our library only contains those algorithms which have been essential for implementing the certifier, whereas Lammich also formalized algorithms for union and complement and thereby proves that regular tree languages are closed under Boolean operations. However, Lammich does neither support ε-transitions nor algorithms which are related to reachability analysis for TRSs.

8. Conclusion

We have introduced the class of deterministic, state-coherent automata that are state-compatible with a TRS $\mathcal{R}$. We have shown that these automata capture precisely those regular tree languages that are closed under rewriting by $\mathcal{R}$, leading to a decision procedure for checking whether a regular language is closed under rewriting. Their simple definition allowed us to formalize most of our results on state-coherent, state-compatible automata within Isabelle/HOL. Also criteria for match-bounds, namely raise-consistency to ensure closure under $\overset{\geqslant}{\to}_{\mathcal{R}'}$, could easily be adapted to the corresponding notion state-raise-consistency. We further demonstrated via examples that the gain in precision carries over to the applications: our notions allows more powerful confluence and termination analysis tools, however it remains as future work to integrate our notions in those search algorithms of the tools which synthesize tree automata.

Further future work consists in the expansion of the formalization w.r.t. match-bounds, e.g., by integrating results on match-bounds for dependency pairs or using match-bounds of right-hand sides of forward closures. Moreover, we plan to generate witnesses of the decision procedure for closure under rewriting in the negative case. Another open question is whether the state-raise-consistency condition can be relaxed to cover more systems.

Acknowledgments

We would like to thank Aart Middeldorp for fruitful discussions on the topic of tree automata and helpful feedback. We are also grateful to the reviewers of the preliminary versions of this paper for their constructive feedback.

References

[1] F. Baader, T. Nipkow, Term Rewriting and All That, Cambridge University Press, 1998.
[2] B. Boyer, T. Genet, T.P. Jensen, Certifying a tree automata completion checker, in: Proc. 4th International Joint Conference on Automated Reasoning, in: Lect. Notes Comput. Sci., vol. 5195, 2008, pp. 523–538.
[3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, M. Tommasi, Tree automata techniques and applications, available at http://tata.gforge.inria.fr, 2007.
[4] J. Endrullis, R.C. de Vrijer, J. Waldmann, Local termination, in: Proc. 20th International Conference on Rewriting Techniques and Applications, in: Lect. Notes Comput. Sci., vol. 5595, 2009, pp. 270–284.
[5] B. Felgenhauer, R. Thiemann, Reachability analysis with state-compatible automata, in: Proc. 8th International Conference on Language and Automata Theory and Applications, in: Lect. Notes Comput. Sci., vol. 8370, 2014, pp. 347–359.
[6] G. Feuillade, T. Genet, V.V.T. Tong, Reachability analysis over term rewriting systems, J. Autom. Reason. 33 (2004) 341–383.
[7] T. Genet, Decidable approximations of sets of descendants and sets of normal forms, in: Proc. 9th International Conference on Rewriting Techniques and Applications, in: Lect. Notes Comput. Sci., vol. 1379, 1998, pp. 151–165.
[8] T. Genet, Y.-M. Tang-Talpin, V.V.T. Tong, Verification of copy-protection cryptographic protocol using approximations of term rewriting systems, in: Proc. WITS'03 (Workshop on Issues in the Theory of Security), 2003.
[9] A. Geser, D. Hofbauer, J. Waldmann, H. Zantema, On tree automata that certify termination of left-linear term rewriting systems, Inf. Comput. 205 (4) (2007) 512–534.
[10] F. Haftmann, T. Nipkow, Code generation via higher-order rewrite systems, in: Proc. 10th International Symposium on Functional and Logic Programming, in: Lect. Notes Comput. Sci., vol. 6009, 2010, pp. 103–117.
[11] N. Hirokawa, A. Middeldorp, Tyrolean termination tool, in: Proc. 16th International Conference on Rewriting Techniques and Applications, in: Lect. Notes Comput. Sci., vol. 3467, 2005, pp. 175–184.
[12] M. Korp, Termination analysis by tree automata completion, Ph.D. thesis, University of Innsbruck, 2010.
[13] M. Korp, A. Middeldorp, Match-bounds revisited, Inf. Comput. 207 (11) (2009) 1259–1283.
[14] Lammich, P., Tree automata. Archive of Formal Proofs, http://afp.sf.net/entries/Tree-Automata.shtml, Formal proof development, Nov. 2009.

[15] P. Lammich, A. Lochbihler, The Isabelle collections framework, in: Proc. 1st International Conference on Interactive Theorem Proving, in: Lect. Notes Comput. Sci., vol. 6172, 2010, pp. 339–354.
[16] A. Lochbihler, Light-weight containers for Isabelle: efficient, extensible, nestable, in: Proc. 4th International Conference on Interactive Theorem Proving, in: Lect. Notes Comput. Sci., vol. 7998, 2013, pp. 116–132.
[17] T. Nipkow, L. Paulson, M. Wenzel, Isabelle/HOL – a Proof Assistant for Higher-Order Logic, Lect. Notes Comput. Sci., vol. 2283, Springer, 2002.
[18] R. Thiemann, C. Sternagel, Certification of termination proofs using CeTA, in: Proc. 22nd International Conference on Theorem Proving in Higher Order Logics, in: Lect. Notes Comput. Sci., vol. 5674, 2009, pp. 452–468.
[19] H. Zankl, B. Felgenhauer, A. Middeldorp, CSI – a confluence tool, in: Proc. 23rd International Conference on Automated Deduction, in: Lect. Notes Artif. Intell., vol. 6803, 2011, pp. 499–505.