SAT-Based Automated Completion for Reachability Analysis

Yohan Boichut[1], Vincent Hugot[12], Adrien Boiret[12]

[1] LIFO, UR 4022 – Université d'Orléans
yohan.boichut@univ-orleans.fr
[2] INSA Centre Val de Loire
first.last@insa-cvl.fr

Abstract. Reachability analysis in rewriting has served as a verification technique in recent decades, despite the underlying issue being undecidable. Regular tree model-checking has found application in verifying security protocols, Java programs, and concurrent systems. The premise in these approaches is to represent the targeted system as a state system and encode its transitions using a term rewriting system or a tree transducer. The crucial aspect lies in calculating a fixed point that represents the set of configurations or states that can be reached. While this is generally uncomputable, it is sufficient to compute an overapproximation for the purpose of verifying safety properties.
Let A, B, and $\mathcal{R}$ represent, respectively, an initial set of terms, a set of forbidden ("bad") terms, and a term rewriting system. The question is whether there exists a regular approximation $A^\star$ of the set of reachable terms such that $A^\star \supseteq \mathcal{R}^*(A)$ and $A^\star \cap B = \emptyset$.
Finding suitable approximations requires, in practice, the use of heuristics, steered towards an anticipated conclusive fixed point by the intervention of human domain experts. The parameters upon which they act may take the form of term equations, normalizing rules, predicate abstractions, etc, but in all cases boil down to carefully choosing states to merge during the fixpoint computation, forcing convergence while avoiding overshooting the approximation into B.
We propose a practical, scalable automated method offloading that expert work to a SAT solver.

1 Introduction

Over the past few decades, the verification of critical systems has been the subject of intense study. Various formal methods, such as model-checking, symbolic model-checking, and theorem proving, have been applied extensively.

Theorem proving techniques often rely on human interaction to prove safety properties. For finite state systems, model-checking has been successfully employed, with its main limitation being the size of the state space. However, verifying infinite state systems using symbolic and regular model-checking poses challenges for automatic verification. The overarching issue of system verification is undecidable. In *regular model-checking* [18], the problem of system verification is transformed into a reachability problem within term rewriting or term transduction.

Given two tree automata (TA) $\mathcal{A}$ and $\mathcal{B}$, representing respectively the initial and the forbidden languages, and a TRS $\mathcal{R}$, the *tree automata completion algorithm* strives to generate a TA $\mathcal{A}^\star$ of language closed under $\mathcal{R}$ and not containing any forbidden term. It does this by adding new rules and states to $\mathcal{A}$. There is no guarantee of termination, however.

Consider the TRS $\mathcal{R}$ composed of a single rule $f(x) \to f(a(x))$ and a TA $\mathcal{A}$ that accepts only the ground term $f(\bot)$. Then $\mathcal{A}^\star$ must accept all terms of the form $f(a^*(\bot))$. While this language is regular, without expert intervention to guide the reuse or merging of new states during the computation of $\mathcal{A}^\star$, it would diverge, adding a new state to recognize each term $f(a^n(\bot))$ (cf. Example 18$_{[p5]}$).

The key parameter which the expert must manipulate can be conceptualized as an equational theory. In practice, it may take various forms, including priority transitions & approximation rules [15], approximation functions [8], sets of term equations, e.g. $a(x) = x$ [9, 11], etc. These approaches have shown effectiveness in verifying security protocols, Java programs or functional programs ([13, 4, 5, 12]).

Automating this technique has been a challenging pursuit that has motivated us for several years. While some solutions have been explored in [1], they are specific to certain application fields, such as security protocols. Other solutions presented in [12] focus on verifying safety properties of higher-order functions. One significant limitation of [12] is that the term rewriting system must be terminating.

The workshop paper [2] made an initial attempt to automate the reachability problem in its broadest sense: given $A, B, \mathcal{R}$ as above, can mergings of states be computed for A such that the resulting language both avoids forbidden terms and is closed under $\mathcal{R}$? This was automated by converting the question into a WS1S formula and feeding it to the MONA [19] solver. The key limitations of [2] are that the language of B must be finite, and that the approach did not scale at all: the cutoff point was ~20 variables.

This paper expands on [2]: we greatly simplify the formalisation — correcting various errors in the process — and remove the key limitation. On the practical side, we target classical propositional logic to leverage the power of SAT solvers, which are highly optimised. As we shall see, this pays off in the experimental results.

Section 2 fixes definitions and notations regarding tree automata and rewriting. Section 3 presents a characterisation of conclusive approximations in propositional logic. In Section 4 integrates the method in the completion process. Section 5 analyses complexity and experimental results.

2 Notations and reminders about tree automata

Given a set S and an equivalence relation $\approx$ on S, we write $S/_{\approx}$ the quotient set (set of equivalence classes), and $[x]_{\approx}$ or $[x]$ the equivalence class of $x \in S$ in S wrt. $\approx$.

Let $\mathbb{X}$ be a finite set of variables and Σ a finite ranked alphabet, where the arity of each symbol $f \in \Sigma$ is denoted by $\mathrm{ar}(f)$. The set of **terms** built from Σ and $\mathbb{X}$ is denoted by $\mathcal{T}(\Sigma, \mathbb{X})$. The set of **ground terms** (without variables) is written $\mathcal{T}(\Sigma)$. The notions of **position**, and **substitution** are defined as usual.

For a term $t \in \mathcal{T}(\Sigma, \mathbb{X})$, $\mathbb{X}(t)$ is the set of variables appearing in t, $\mathcal{P}(t)$ is the set of positions of t, as words of $[\![0, n]\!]^*$. Thus, a term is seen as a function $t \in (\mathcal{P}(t) \to \Sigma \cup \mathbb{X})$. It is **linear** if each variable of $\mathbb{X}(t)$ appears at exactly one position.

A **rewrite rule** is a pair of terms, written $l \to r$, such that $\mathbb{X}(l) \supseteq \mathbb{X}(r)$ and $l \notin \mathbb{X}$. It is **left-** (resp. **right-**)**linear** if l (resp. r) is linear, and just **linear** if it is both left- and right-linear. A **term rewriting system (TRS)** $\mathcal{R}$ is a finite set of rewrite rules. It is **left-** (resp. **right-**)**linear** if all its rules are. It induces a rewrite relation $\to_{\mathcal{R}}$ on $\mathcal{T}(\Sigma)$, whose reflexive and transitive closure is written $\to_{\mathcal{R}}^*$. We let $\mathcal{R}^*(L) = \{ v \mid u \in L \wedge u \to_{\mathcal{R}}^* v \}$.

Let $\mathbb{Q}$ be a countably infinite set of elements called **states**, all of arity 0, and such that $\mathbb{X}$, $\mathbb{Q}$, and Σ are all pairwise disjoint. A **configuration** is a (ground) term $c \in \mathcal{T}(\Sigma \cup \mathbb{Q})$. A **transition** is a rewrite rule $c \to q \in \mathcal{T}(\Sigma \cup \mathbb{Q}) \times \mathbb{Q}$; it is **normal** if either $c \in \mathbb{Q}$ (in which case it is an **ε-transition**) or $c = f(q_1, \dots, q_n)$, with $q_k \in \mathbb{Q}, \ \forall k$.

Definition 1. A **nondeterministic bottom-up tree automaton (TA)** A is a tuple $\langle \Sigma, Q, F, \Delta \rangle$, where Σ is a ranked alphabet, $Q \subseteq \mathbb{Q}$ is a set of states, $F \subseteq Q$ is a set of final states, and $\Delta \subseteq \mathcal{T}(\Sigma, Q) \times Q$ is a set of normal transitions. It is **ε-free** if it has no ε-transition, which we shall safely assume whenever convenient [6, p23].

Given a state $q \in Q$, its **language in A** is defined as $[\![q]\!]_A = \{ t \in \mathcal{T}(\Sigma) \mid t \to_{\Delta}^* q \}$. We write $[\![q]\!]$ when A is clear from context. The **language of A** is $[\![A]\!] = \bigcup_{q \in F} [\![q]\!]$. We often write $\to_A$ for $\to_\Delta$. The **synchronised product of two TA, $A \times B$**, is defined as usual [6, p30].

By convention, any TA A will be assumed to be defined as above. We employ object-like notations: e.g. $B.Q$ is the set of states of B, and $B = \langle \Sigma := f(\Sigma), \ Q := \dots \rangle$ defines a new automaton whose alphabet is a function of Σ (meaning $A.\Sigma$ in context, given the definition of A above), etc. We can define a new automaton from another, keeping most attributes implicitly unaltered, but automatically updating them if needed; for instance, $B = A\langle \Delta := \Delta \cup \{ f(p) \to q \} \rangle$ defines B to be the same as A, but with the addition of rule $f(p) \to q$. It is thus shorthand for $B = \langle \Sigma := \Sigma, Q := Q \cup \{ p, q \}, \ F := F, \ \Delta := \Delta \cup \{ f(p) \to q \} \rangle$.

Given a TA A and a TRS $\mathcal{R}$, a **critical pair** is a tuple (σ, l, r, q) [(c)] where $\sigma : \mathbb{X} \to Q$ is a substitution and $l \to r \in \mathcal{R}$ is a rewrite rule, such that there is a state q with $\sigma l \to_A^* q$ and $\sigma r \not\to_A^* q$. We denote by $\mathfrak{C}^{\mathrm{rit}}_{\mathcal{R},A}$ the set of critical pairs:

$$\mathfrak{C}^{\mathrm{rit}}_{\mathcal{R},A} \quad = \quad \left\{ (\sigma, l, r, q) \;\middle|\; l \to r \in \mathcal{R}, \ \sigma l \to_A^* q, \ \sigma r \not\to_A^* q \right\} . \qquad (\mathfrak{C}^{\mathrm{rit}}_{\mathcal{R},A})$$

(c) This is usually presented as the pair $(\sigma l, \sigma r)$, hence the name "critical pair".

$\mathcal{A}$ is said to be **acritical** (wrt. $\mathcal{R}$) if there are no critical pairs. If so, $\mathcal{A}$ is also **complete**, that is to say $\mathcal{R}^*(\llbracket\mathcal{A}\rrbracket) \subseteq \llbracket\mathcal{A}\rrbracket$. The converse is not true.

A **completion step** consists in solving all critical pairs by adding transitions – in dotted line – such that

$$\begin{tikzcd} \sigma l \arrow[r, "\mathcal{R}"] \arrow[d, "\mathcal{A}"', "*"] & \sigma r \arrow[dl, dashed, bend left, "\mathcal{A}", "*"'] \\ q & \end{tikzcd} .$$

3 SAT solving for overapproximations

Let $\mathcal{A}$ and $\mathcal{B}$ be TA and $\mathcal{R}$ a TRS. Here $\mathcal{A}$ can be the initial language, or may already be extended by one or several steps of completion. In any case, the pivotal issue is testing $\mathcal{R}^*(\llbracket\mathcal{A}\rrbracket) \cap \llbracket\mathcal{B}\rrbracket = \varnothing$. While this question is undecidable, a well-chosen over-approximation of $\mathcal{R}^*(\llbracket\mathcal{A}\rrbracket)$ can suffice to conclude. In this section, we give a logical characterisation of whether such an approximation can be obtained purely by merging states of $\mathcal{A}$ — and, if so, how.

Definition 2. Given a TA $\mathcal{A}$ and an equivalence relation $\approx$ on $\mathcal{Q}$, which we call simply an **approximation**, the **quotient automaton** ${}^{\mathcal{A}}\!/\!_{\approx}$ is defined as usual:

$$\mathcal{A}/_{\approx} \;=\; \left\langle \begin{array}{l} \Sigma := \Sigma \qquad \mathcal{Q} := \mathcal{Q}/_{\approx} \qquad \mathcal{F} := \left\{ [q]_{\approx} \;\middle|\; q \in \mathcal{F} \right\} \\ \Delta := \left\{ f(\dots[q_i]_{\approx}\dots) \to [q]_{\approx} \;\middle|\; f(\dots q_i \dots) \to q \in \Delta \right\} \end{array} \right\rangle \tag{1}$$

Note that we have obviously $\llbracket\mathcal{A}\rrbracket \subseteq \llbracket{}^{\mathcal{A}}\!/\!_{\approx}\rrbracket$, for any $\mathcal{A}$ and $\approx$.

Definition 3. Given TA $\mathcal{A}$, $\mathcal{B}$ and a TRS $\mathcal{R}$, an approximation $\approx$ is **acritical** if ${}^{\mathcal{A}}\!/\!_{\approx}$ has no critical pair (which implies that it is also **complete**: $\mathcal{R}^*(\llbracket\mathcal{A}\rrbracket) \subseteq \llbracket{}^{\mathcal{A}}\!/\!_{\approx}\rrbracket$) and **suitable** if $\llbracket{}^{\mathcal{A}}\!/\!_{\approx}\rrbracket \cap \llbracket\mathcal{B}\rrbracket = \varnothing$. It is **conclusive** if it is both acritical and suitable.

Problem 4 (States Merging). Given TA $\mathcal{A}$, $\mathcal{B}$ and a left-linear [(d)] TRS $\mathcal{R}$, find (if it exists) a conclusive approximation $\approx$, or prove its non-existence.

In all that follows, $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{R}$ are defined as in Problem 4.

We reduce that problem to SAT by building a formula of propositional logic whose valuation describes $\approx$. The **relevant variables** of that formula are $\mathbb{V} = \{\, p \approx q \mid p, q, \in \mathcal{Q} \,\}$. We call such a formula φ an **approximation formula**.

The semantics of approximation formulæ are defined as usual:

$$\llbracket\varphi\rrbracket \;=\; \left\{ V \;\middle|\; V \models \varphi \right\} . \tag{2}$$

A valuation V defines a relation $\operatorname{rel} V$ (V is an **underlying valuation** of $\operatorname{rel} V$):

$$\operatorname{rel} V \;=\; \left\{ (p, q) \;\middle|\; V(p \approx q) = \top \right\} . \tag{3}$$

[(d)] This the usual restriction for TA completion. See [14, Sec. 4.4.1] for a discussion.

Thus, a formula φ defines a set of relations:

$$\{\!|\varphi|\!\} \;=\; \left\{ \operatorname{rel} V \;\middle|\; V \in [\![\varphi]\!] \right\} . \tag{4}$$

Now we need to ensure that those relations are **(1)** approximations that are **(2)** suitable and **(3)** acritical.

To ensure that those atoms of the form "$p \approx q$" do indeed describe an equivalence relation, and therefore an approximation, we start by enforcing reflexivity, symmetry, and transitivity.

Definition 5 (φ^e)**.** We denote by φ^e_A, or simply φ^e, the **equivalence formula** defined as:

$$\varphi^e_A \;=\; \bigwedge_{p\in Q} p \approx p \;\wedge\; \bigwedge_{\substack{p,q\in Q\\ p<q}} p \approx q \Leftrightarrow q \approx p \;\wedge\; \bigwedge_{\substack{p,q,r\in Q\\ p\neq q\neq r\\ p<r}} (p \approx q \wedge q \approx r) \Rightarrow p \approx r \,. \tag{5}$$

The conditions $p < q$ and $p \neq q \neq r \wedge p < r$ in φ^e are trivial simplifications that can be made without loss of generality given any arbitrary order on Q to generate smaller formulæ. In the implementation, atoms $p \approx p$ are never actually generated in any formula, but directly replaced by $\top$.

Proposition 6 *(Equivalence).* $\{\!|\varphi^e|\!\}$ *is the set of all approximations.*

To enforce suitability, we translate, rather straightforwardly, the non-existence of any accepting run for $A \times B$.

Definition 7 (φ^s)**.** Let $P = A \times B$. We denote by $\boldsymbol{\varphi^s_{\mathcal{R},A,B}}$, or simply φ^s, the **suitability formula** on variables $P.Q \cup \mathbb{V}$, defined as:

$$\varphi^s_{\mathcal{R},A,B} \;=\; \overbrace{\left[\neg \bigvee_{X\in P.F} X\right]}^{\varphi^s_i} \;\wedge\; \overbrace{\bigwedge_{(p,q)\in P.Q} \left[(p,q) \iff \bigvee_{\substack{f(\dots,(p_i,q_i),\dots)\to(p',q)\\ \in P.\Delta}} p \approx p' \;\wedge\; \bigwedge_i (p_i, q_i) \right]}^{\varphi^s_r} \tag{6}$$

The (p, q) variables are not *relevant* in the sense defined above, and we shall ignore their valuation when extracting the approximation, but they serve as "scaffolding" for the formula. Without them, this would be longer and much more difficult. They are used only in φ^s.

Proposition 8 *(Suitability).* $\{\!|\varphi^e \wedge \varphi^s|\!\}$ *is the set of all suitable approximations.*

Proof. **⊆ suitable.**

Let $\approx \in \{\!|\varphi^e \wedge \varphi^s|\!\} \subseteq \{\!|\varphi^e|\!\}$. By Proposition 6, it is an approximation. We show it is suitable.

Let us start by showing that, if the state $([p]_\approx, q)$ is accessible in ${}^{A}\!/\!_{\approx} \times B$ then the variable (p, q) is true in any underlying valuation; this is done by structural induction on trees of $[\![([p]_\approx, q)]\!]$.

- *Base case:* $t = a$*:*
  If $a \in \llbracket([p]_{\approx}, q)\rrbracket$ then $a \to [p]_{\approx} \in {}^{A}\!/\!_{\approx}.\Delta$ and $a \to q \in B.\Delta$, thus there exists p' such that $p' \approx p$ and $a \to p' \in A.\Delta$. Therefore $a \to (p', q) \in P.\Delta$, and that rule satisfies the disjunction on the rules of $P.\Delta$ of φ^s_r, which means that (p, q) is true.
- *Inductive case:* $t = f(t_1, \ldots, t_n)$*:*
  If $t \in \llbracket([p]_{\approx}, q)\rrbracket$ then $\forall i \in \llbracket 1, n\rrbracket$ we have p_i, q_i such that:
  (1) $f([p_1]_{\approx}, \ldots, [p_n]_{\approx}) \to [p]_{\approx} \in {}^{A}\!/\!_{\approx}.\Delta$, **(2)** $f(q_1, \ldots, q_n) \to q \in B.\Delta$, **(3)** $t_i \in \llbracket([p_i]_{\approx},$
  By the first point there must exist $p', p'_1, \ldots, p'_n$ such that $p' \approx p$, all $p'_i \approx p_i$, and $f(p'_1, \ldots, p'_n) \to p' \in A.\Delta$. Combined with $f(q_1, \ldots, q_n) \to q \in B.\Delta$, this gives $f\big((p'_1, q_1), \ldots, (p'_n, q_n)\big) \to (p', q) \in P.\Delta$. By $t_i \in \llbracket([p_i]_{\approx}, q_i)\rrbracket$ and $p'_i \approx p_i$ we also have that $t_i \in \llbracket([p'_i]_{\approx}, q_i)\rrbracket$, which by induction means that we can assume (p'_i, q_i) to be true. The rule $f((p'_1, q_1), \ldots, (p'_n, q_n)) \to (p', q)$ satisfies the disjunction on rules of $P.\Delta$ and the truth of (p'_i, q_i) satisfies the conjunction on i of φ^s_r, which means that (p, q) is true.

Now, by φ^s_l, no $(p, q) \in P.F$ is true, so no $([p]_{\approx}, q) \in ({}^{A}\!/\!_{\approx} \times B).F$ can be accessible, thus $\llbracket {}^{A}\!/\!_{\approx} \rrbracket \cap \llbracket B \rrbracket = \varnothing$, and $\approx$ is suitable.

$\supseteq$ suitable.

Let $\approx$ be a suitable approximation, let us show that $\approx \in \{\!|\varphi^s|\!\}$. We'll complete the underlying valuation of $\approx$ so that (p, q) is true iff $\llbracket([p]_{\approx}, q)\rrbracket \neq \varnothing$, and show this satisfies φ^s.

First, φ^s_l. Since $\approx$ is suitable, we have $\llbracket {}^{A}\!/\!_{\approx} \rrbracket \cap \llbracket B \rrbracket = \varnothing$, thus there can be no $(p, q) \in (A \times B).F$ such that $\llbracket([p]_{\approx}, q)\rrbracket \neq \varnothing$. Hence, all states of $P.F$ are set to false, and φ^s is satisfied.

Second, φ^s_r. By our definition (p, q) is true iff there exists a term $t = f(t_1, \ldots, t_n) \in \llbracket([p]_{\approx}, q)\rrbracket$. This is to say that t is evaluated by rules of the form $f(\ldots, [p_i]_{\approx}, \ldots) \to [p]_{\approx} \in {}^{A}\!/\!_{\approx}.\Delta$ and $f(\ldots, q_i, \ldots) \to q \in B.\Delta$. Equivalently, **(1)** there is a rule $f(\ldots, (p_i, q_i), \ldots) \to (p', q) \in P.\Delta$, **(2)** for all i, $t_i \in \llbracket([p_i]_{\approx}, q_i)\rrbracket$, which means all (p_i, q_i) are set to true, and **(3)** $p' \approx p$. Thus (p, q) is true iff the matching disjunctive clause is true; φ^s_r is satisfied. □

There remains to enforce acriticality. We are going to translate "under the approximation, there are no critical pairs".

To do so, we need to compute under which approximations a term can be evaluated (i.e. rewritten) into a given state; this will enable us to enforce, for all rules $l \to r \in \mathcal{R}$, that if a term t matching l can be evaluated in q under some approximation, then there exists a compatible approximation under which the rewritten term can also be evaluated in q, thus enforcing acriticality.

For this, we shall implement a notion of *unifiers*, $\mathcal{U}^t_q$, giving us suitable formulæ and substitutions under which t evaluates in q.

Definition 9 (Unifier)**.** Let $t \in \mathcal{T}(\Sigma, Q \cup \mathbb{X})$ be a term linear in $\mathbb{X}$, and let $q \in Q$. The set of **unifiers** of t and q, written $\mathcal{U}^t_q$, is defined as:

$$\mathcal{U}^t_q = \begin{cases} \{\, (\top,\ \{t \mapsto q\}) \,\} & \text{if} \quad t \in \mathbb{X} \\ \{\, (t \approx q,\ \varnothing) \,\} & \text{if} \quad t \in Q \\ \displaystyle\bigcup_{f(q_1,\ldots,q_n)\to p \in \Delta} \{\, (p \approx q,\ \varnothing) \,\} \otimes \bigotimes_{k=1}^{n} \mathcal{U}^{t_k}_{q_k} & \text{if} \quad t = f(t_1,\ldots,t_n) \end{cases}, \tag{7}$$

where $(\varphi, \sigma) \otimes (\psi, \rho) = (\varphi \wedge \psi,\ \sigma \cup \rho)$, neutral element $(\top, \varnothing)$, and

$$\{\ldots, (\varphi_i, \sigma_i), \ldots\} \otimes \{\ldots, (\psi_j, \rho_j), \ldots\} = \{\ldots, (\varphi_i, \sigma_i) \otimes (\psi_j, \rho_j), \ldots\}\,. \tag{8}$$

Note that left-linearity of $\mathcal{R}$ justifies the functionality of $\sigma \cup \rho$.

It may seem a bit unprincipled to have $\mathcal{U}^t_q$ operate "modulo $\approx$" in all cases but $t \in \mathbb{X}$. A more natural definition would certainly be $\{\, (p \approx q,\ \{t \mapsto p\}) \mid p \in Q \,\}$. We generally opted, for this paper, to favour simplicity and directness in the formulæ and proofs over performance, and to reserve discussion of optimisations to further works. This is the exception. The practical performance impact of that single choice would be colossal, reducing the scope of the method by several orders of magnitude in most cases. The cost of choosing the computationally cheaper $\{\, (\top,\ \{t \mapsto q\}) \,\}$ is paid in the remaining proofs: in many instances we need to explicitly reason "modulo $\approx$", because the unifier won't do it for us. To do that, given $\sigma, \sigma' : \mathbb{X} \to \mathbb{Q}$, we write $\boldsymbol{\sigma \approx \sigma'}$ if $\forall x,\ \sigma(x) \approx \sigma'(x)$.

We consider ${}^{\mathcal{A}}\!/\!_{\approx}$ for some given $\approx$. We extend the definition of $[\cdot]_\approx$ from states of Q to configurations $\mathcal{T}(\Sigma \cup Q)$ inductively: $[f(t_1, \ldots, t_n)]_\approx = f([t_1]_\approx, \ldots, [t_n]_\approx)$. Put another way, $[\cdot]_\approx$ lifts configurations of $\mathcal{A}$ to corresponding configurations of ${}^{\mathcal{A}}\!/\!_{\approx}$. We write $[\cdot]_\approx$ simply as $[\cdot]$, since $\approx$ is the same throughout our proofs.

We now offer a characterisation of unifiers.

Lemma 10 (*Unifier characterisation*)**.** *Let* ${}^{\mathcal{A}}\!/\!_{\approx}$ *be an* ε*-free quotient automaton,* t *a term of* $\mathcal{T}(\Sigma \cup Q, \mathbb{X})$ *linear in* $\mathbb{X}$*, and* $\sigma : \mathbb{X}(t) \to Q$ *a substitution. Then* $[\sigma t] \to^*_{\mathcal{A}/\approx} [q]$ *iff there exists* $(\varphi, \sigma') \in \mathcal{U}^t_q$ *such that* $\approx\, \in \{\!\!\{\varphi\}\!\!\}$ *and* $\sigma' \approx \sigma$.

Proof. This can be proven by induction on t.

Case $t = p \in Q$: We have $\sigma = \varnothing$ and $[p] \to^*_{\mathcal{A}/\approx} [q]$ iff $[p] = [q]$ iff $p \approx q$. Since $\mathcal{U}^p_q = \{\, (t \approx q,\ \varnothing) \,\}$, the equivalence holds.

Case $t = x \in \mathbb{X}$: We have $\sigma = \{\, x \mapsto p \,\}$ for some state p, and by definition $\mathcal{U}^x_q = \{\, (\top, \sigma' := \{\, x \mapsto q \,\}) \,\}$. Then $[\sigma t] = [p] \to^*_{\mathcal{A}/\approx} [q]$ iff $p \approx q$ iff $\sigma' \approx \sigma$.

Case $t = f(t_1, \dots, t_n)$**:** There are σ_i such that $\sigma t = f(\dots, \sigma_i t_i, \dots)$ and $\sigma = \bigcup \sigma_i$ — the functionality of σ is guaranteed by the linearity of t. We have $[\sigma t] = f(\dots, [\sigma_i t_i], \dots) \rightarrow^*_{A/\approx} [q]$ iff there exists $q_1, \dots, q_n$ such that **(1)** for all i, $[\sigma_i t_i] \rightarrow^*_{A/\approx} [q_i]$ and **(2)** $f(\dots, [q_i]_\approx, \dots) \rightarrow [q]_\approx \in {}^{A}\!/\!_{\approx}.\Delta$ The latter condition can, by definition of ${}^{A}\!/\!_{\approx}$, be replaced by **(2')** $\exists p : f(\dots, q_i, \dots) \rightarrow p \in \Delta \ \wedge \ p \approx q$.

By induction, $[\sigma_i t_i] \rightarrow^*_{A/\approx} [q_i]$ iff $\exists (\varphi_i, \sigma'_i) \in \mathrm{U}^{t_i}_{q_i} : \approx \in \{\!|\varphi_i|\!\} \wedge \sigma'_i \approx \sigma_i$. Thus, by definition of $\otimes$, we can reformulate **(1)** as **(1')** $\exists (\varphi := \bigwedge \varphi_i, \ \sigma' := \bigcup \sigma'_i) \in \bigotimes_{i=1}^{n} \mathrm{U}^{t_i}_{q_i} : \approx \in \{\!|\varphi|\!\} \wedge \sigma' \approx \sigma$.

In the end, $[\sigma t] \rightarrow^*_{A/\approx} [q]$ iff there exists a rule $f(\dots, q_i, \dots) \rightarrow p \in \Delta$, $\sigma' \approx \sigma$ and φ such that $(\varphi, \sigma') \in \bigotimes_{i=1}^{n} \mathrm{U}^{t_i}_{q_i}$ and $\approx \in \{\!|p \approx q \wedge \varphi|\!\}$, that is to say iff there exists φ' such that $(\varphi', \sigma) \in \mathrm{U}^{t}_{q}$. ◻

Definition 11 (φ^a)**.** We denote by $\boldsymbol{\varphi^a_{\mathcal{R},A}}$, or simply φ^a, the **acriticality formula**:

$$\varphi^a_{\mathcal{R},A} \quad = \quad \bigwedge_{\substack{l \to r \in \mathcal{R} \\ q \in Q}} \ \bigwedge_{(\alpha,\sigma) \in \mathrm{U}^l_q} \left[\alpha \ \Rightarrow \bigvee_{(\beta,\varnothing) \in \mathrm{U}^{r\sigma}_q} \beta \right] \tag{9}$$

Proposition 12 (*Acriticality*)**.** $\{\!|\varphi^e \wedge \varphi^a|\!\}$ *is the set of all acritical approximations.*

Proof. Let $\approx \in \{\!|\varphi^e \wedge \varphi^a|\!\}$. By Prop. 6, it is an approximation, and Lemma 10 applies. Since all configurations of ${}^{A}\!/\!_{\approx}$ have representatives in A, the acriticality of ${}^{A}\!/\!_{\approx}$ can be expressed as: for every rule $l \to r \in \mathcal{R}$, state $q \in Q$, and substitution σ, if $[\sigma l] \rightarrow^*_{A/\approx} [q]$ then $[\sigma r] \rightarrow^*_{A/\approx} [q]$.

By Lem 10, this means that for every l, r, q, σ, if there exists $(\alpha, \sigma') \in \mathrm{U}^l_q$ such that $\sigma \approx \sigma'$, and $\approx \in \{\!|\alpha|\!\}$, then there exists $(\beta, \varnothing) \in \mathrm{U}^{\sigma r}_q$ and $\approx \in \{\!|\beta|\!\}$. Since $\sigma' \approx \sigma$, we have, by definition, $\mathrm{U}^{\sigma r}_q = \mathrm{U}^{\sigma' r}_q$. We can wlog. quantify directly on the substitutions actually occurring in the unifiers — here σ'.

By simple reformulation of the quantifiers, this is equivalent to: for every rule $l \to r \in \mathcal{R}$ and state $q \in Q$ (first conjunction of φ^a), for all $(\alpha, \sigma') \in \mathrm{U}^l_q$ (second conjunction of φ^a), if $\approx$ satisfies α, then there exists β such that $(\beta, \varnothing) \in \mathrm{U}^{\sigma' r}_q$, and $\approx$ satisfies β (inner disjunction of φ^a).

Hence, ${}^{A}\!/\!_{\approx}$ is acritical iff $\approx \in \{\!|\varphi^a|\!\}$. ◻

Definition 13. We denote by $\boldsymbol{\varphi_{\mathcal{R},A,B}}$, or simply φ, the **conclusiveness formula**:

$$\varphi_{\mathcal{R},A,B} \quad = \quad \varphi^e \wedge \varphi^s \wedge \varphi^a \tag{10}$$

Theorem 14 (*Conclusiveness*)**.** $\{\!|\varphi_{\mathcal{R},A,B}|\!\}$ *is the set of all conclusive approximations.*

4 The completion algorithm

Given a TA, we can now automatically find a suitable approximation of it by feeding φ to a SAT solver. There remains to integrate this to the completion algorithm.

Section 4.1 recalls the notions and fixes our notations for classical completion. Section 4.2 discusses how to integrate our search for suitable approximations in a completion process.

4.1 Reminders about classical completion

Given two TA $\mathcal{A}$, $\mathcal{B}$ and a TRS $\mathcal{R}$, the tree automata completion algorithm introduced in [10] attempts to compute a fixpoint automaton $\mathcal{A}^\star$ such that $\mathcal{R}^*(⟦\mathcal{A}^\star⟧) \subseteq ⟦\mathcal{A}^\star⟧$ and $⟦\mathcal{A}^\star⟧ \cap ⟦\mathcal{B}⟧ = \varnothing$. The search for $\mathcal{A}^\star$ commences from $\mathcal{A}$ and extends it with new states and rules to recognise more and more rewritten terms via successive completion steps — cf. Eq. $(\mathcal{C}^{crit}_{\mathcal{R},\mathcal{A}})_{[p2]}$ and below. However, σr may be a term of arbitrary depth, and so the new rules $\sigma r \to q$ must be normalised.

Definition 15 (Rule Normalisation)**.** Let $t \to q$ be a transition and $Q \subseteq \mathbb{Q}$ a set of states. The normalisation of $t \to q$ avoiding Q is written $\overline{t \to q}^Q$ and defined as:

$$\overline{f(t_1, \dots, t_n) \to q}^Q = \left\{\, f\left(t_1^!, \dots, t_n^!\right) \to q \,\right\} \cup \bigcup_{i=1}^{n} \overline{t_i \to t_i^!}^Q \tag{11}$$

$$\overline{p \to q}^Q = \{\, p \to q \,\}, \quad \forall p \neq q \in Q \tag{12}$$

$$\overline{q \to q}^Q = \varnothing, \quad \forall q \in Q\,, \tag{13}$$

where, for all i, $t_i^! = t_i$ if $t_i \in Q$, and otherwise $t_i^!$ is some fresh state from $\mathbb{Q} \setminus Q$. The operation is extended to sets of rules in the obvious way.

Definition 16 (Completion operation)**.** We denote by $\mathcal{R}^{\|}(\mathcal{A})$ [(e)] the automaton obtained from $\mathcal{A}$ by **one step of completion** wrt. $\mathcal{R}$. It is defined as:

$$\mathcal{R}^{\|}(\mathcal{A}) \quad = \quad \mathcal{A}\left\langle \Delta := \Delta \cup \overline{\left\{\, \sigma\, r \to q \;\middle|\; (\sigma, l, r, q) \in \mathcal{C}^{crit}_{\mathcal{R},\mathcal{A}} \,\right\}}^Q \right\rangle . \tag{14}$$

Definition 17 (Classical Completion Algorithm)**.** Given TA $\mathcal{A}$ and a TRS $\mathcal{R}$, we let

$$\Gamma_{\mathcal{R}}(\mathcal{A}) = \begin{cases} \mathcal{A} & \text{if } \mathcal{C}^{crit}_{\mathcal{R},\mathcal{A}} = \varnothing \\ \Gamma_{\mathcal{R}}\left(\mathcal{R}^{\|}(\mathcal{A})\right) & \text{otherwise.} \end{cases} \tag{15}$$

The completion algorithm in this pure form will generally not terminate.

Example 18. Let $\mathcal{R} = \left\{\, f(x) \to f(a(x)) \,\right\}$. Let $\mathcal{A}$ be a TA such that $F = \{q_f\}$ and $\Delta = \left\{\, \bot \to q_\bot, f(q_\bot) \to q_f \,\right\}$; we have $⟦\mathcal{A}⟧ = \{\, f(\bot)\,\}$. A first completion step is

(e) Intuitively, a completion step enables $\mathcal{A}$ to recognise all terms obtainable by applying any number of rules of $\mathcal{R}$ *in parallel* — and exactly those terms if $\mathcal{R}$ is right-linear — hence the notation.

then performed as follow:

$$\mathcal{R}^{\|}(\mathcal{A}) \quad = \quad \mathcal{A}\left\langle \Delta := \Delta \cup \overline{f(a(q_\perp)) \to q_f}^{Q} \right\rangle$$

where $\overline{f(a(q_\perp)) \to q_f}^{Q} = \{\, a(q_\perp) \to p_0,\ f(p_0) \to q_f \,\}$. Consequently, performing a second completion step would lead to:

$$\mathcal{R}^{\|}\big(\mathcal{R}^{\|}(\mathcal{A})\big) \quad = \quad \mathcal{R}^{\|}(\mathcal{A})\left\langle \Delta := \Delta \cup \overline{f(a(p_0)) \to q_f}^{Q} \right\rangle.$$

And so on ad infinitum.

This is where human intervention is required: during each completion step the expert must notice such problems and suggest, for instance, to merge p_0 and $q_\perp$ in $\mathcal{R}^{\|}(\mathcal{A})$ (either directly or through more abstract tools; e.g. in [9] the expert would give the equation $a(x) = x$) making $\mathcal{R}^{\|}(\mathcal{A})$ acritical and yielding the expected language $f(a^*(\perp))$ — which in this example happens to be $\mathcal{R}^*(\llbracket \mathcal{A} \rrbracket)$ exactly. Of course the expert would also take care to avoid $\mathcal{B}$; for instance merging q_f with the others is unsuitable if $\llbracket \mathcal{B} \rrbracket = \{\perp\}$.

Our goal is of course to automate this manual step, replacing the human expert by a SAT solver.

4.2 Completion with automated state merging, and variants

The modified algorithm is basically the same as before, but the termination criterion is not whether the automaton is acritical, but whether it can be conclusively approximated.

Definition 19 (Automated Approximate Completion Semi-Algorithm)**.** Given TA $\mathcal{A}$, $\mathcal{B}$ and a TRS $\mathcal{R}$, we let

$$\Gamma^{\mathrm{aut}}_{\mathcal{R},\mathcal{B}}(\mathcal{A}) = \begin{cases} \mathcal{A}/_{\approx} & \text{if} \approx \in \{\!\!\{\varphi_{\mathcal{R},\mathcal{A},\mathcal{B}}\}\!\!\} \\ \Gamma^{\mathrm{aut}}_{\mathcal{R},\mathcal{B}}\big(\mathcal{R}^{\|}(\mathcal{A})\big) & \text{otherwise.} \end{cases} \tag{16}$$

In cases where several conclusive approximations $\approx$ are available, we select one arbitrarily. Whether $\exists \approx \in \{\!\!\{\varphi_{\mathcal{R},\mathcal{A},\mathcal{B}}\}\!\!\}$ is of course tested using a SAT solver. Some obvious optimisations can be made — for instance, the "live-states" simplification: starting from the first completion, there is no use trying to merge the states of the original $\mathcal{A}$ directly, which shortens the formula a bit.

Let's apply this to Example 18, for $\llbracket \mathcal{B} \rrbracket = \{\perp\}$, with live-states simplification in the unifiers. We have, for φ^a, once all $\top$ and $\perp$ are simplified away:

$$\begin{aligned} &(p_0 \approx q_f \Rightarrow ((p_0 \approx q_f \wedge q_\perp \approx p_0) \vee p_0 \approx q_f)) \\ &\wedge (p_0 \approx q_f \Rightarrow ((p_0 \approx q_f \wedge q_\perp \approx p_0 \wedge q_\perp \approx p_0) \vee (p_0 \approx q_f \wedge q_\perp \approx p_0))) \\ &\qquad\qquad\qquad\qquad\qquad\qquad \wedge ((q_\perp \approx p_0 \wedge q_\perp \approx p_0) \vee q_\perp \approx p_0) \end{aligned}$$

We also have $\varphi^s = \neg\langle q_f, p_0\rangle \wedge \langle q_\perp, p_0\rangle \wedge (\langle q_f, p_0\rangle \iff q_f \approx q_\perp))$. Note that the seemingly unconditional conjuncts come from $\top\perp$-simplifications, for instance $\langle q_\perp, p_0\rangle$ was originally $\langle q_\perp, p_0\rangle \Leftrightarrow (\top \wedge \top)$.

This gives us two possible solutions: $q_\perp \approx p_0$ and $q_\perp \approx p_0 \approx q_f$. Of these, only the first is suitable.

The purpose of performing completion steps is the same as usual: the starting automaton may lack the structure to capture $\mathcal{R}$, regardless of approximation. In Example 18, A could not at all handle the symbol a before completion. While we think this is the most *practical* way of integrating State Merging into completion, there are other ways to do so.

Whereas the classical approach involves expert intervention at each completion step, performing mergings that carry on to the next step, (with possibility of backtracking), this algorithm performs just one merge at the end. This is a good heuristic, but is less general than the fully interactive approach in that, in some cases, even if a conclusive approximation exists, we are not guaranteed to find it. Consider

$$\mathcal{R} = \left\{ \begin{array}{l} f(x,y) \to f\big(s(x), s(y)\big),\ f\big(s(x), s(y)\big) \to f(x,y), \\ f\big(0, s(x)\big) \to a,\ f\big(s(x), 0\big) \to a, \end{array} \right\} . \tag{17}$$

Let A be a TA such that $[\![A]\!] = \{\, f(0,0)\,\}$. Reachability analysis searches for, and can theoretically reach [11], given the right expert choices for state mergings after each completion step, any $\mathcal{R}$-closed regular over-approximation L of $[\![A]\!]$. We know by [3] that in all cases, we must have $a \in L$. A valid example would be

$$L = \left\{ f\big(s^n(0), s^m(0)\big) \;\middle|\; n, m \geqslant 0 \right\} \cup \{\, a \,\} . \tag{18}$$

Yet, this language cannot be found by performing just one final state merging in a fully automatic application of Def. 19, no matter how many completion steps are done beforehand.

Indeed, since $\mathcal{R}$ is linear and we do not manually merge states at each step of Def. 19, the completion by itself cannot introduce any approximation, but merely adds rules to recognise, exactly, new terms of the form $f\big(s^n(0), s^n(0)\big)$ ad infinitum, exactly as in Ex. $18_{[\text{p5}]}$.

Thus, at no point of the process can a state merging generate L, because rules like $f\big(0, s(x)\big)$ can never be evaluated by the automaton, and thus the automaton cannot contain any transition of the form $a \to q$. Those can only be introduced by solving a critical pair *after* a merging has made unbalanced terms recognisable, which cannot occur in Def. 19.

Certainly, we could develop a semi-algorithm that alternates between rewriting steps and state merging, testing all possibilities or backtracking when needed, but this is impractical — equivalent to enumerating all TA.

An intermediate approach would be that of [2, Algo. 6.1]: not performing any actual completion, instead adding at each step, for each $l \to r \in \mathcal{R}$, normalised rules $\{\dots, x_i \mapsto q_i, \dots\}r \to q$ for fresh q_i, q. This adds the necessary structure to the automaton, leaving the task of actually making the fresh states reachable to the State Merging problem. However, this can introduce a lot of redundant structure, making the approach costly compared to the heuristic of Def. 19.

5 Complexity, implementation and tests

How scalable can this approach be in practice? Both φ^e and φ^s are clearly polynomial (cubic and quadratic, respectively), so the critical question is the behaviour of φ^a.

Proposition 20 (*Worst-Case*)**.** $|\varphi^a_{\mathcal{R},A,B}| = O\big(|Q| \times |\mathcal{R}| \times N \times |\Delta|^N\big)$, *where* $N = \max_{r \in \mathcal{R}} |r|$.

Proof. The size of a rule is defined as $|l \to r| = |l| + |r|$, and the size of an element of unifier is defined a $|(\alpha, \sigma)| = |\alpha| + |\sigma|$.

The size of each formula of U^t_q is exactly $|t|$, and $\left|U^t_q\right| \leqslant |\Delta|^{|t|}$.

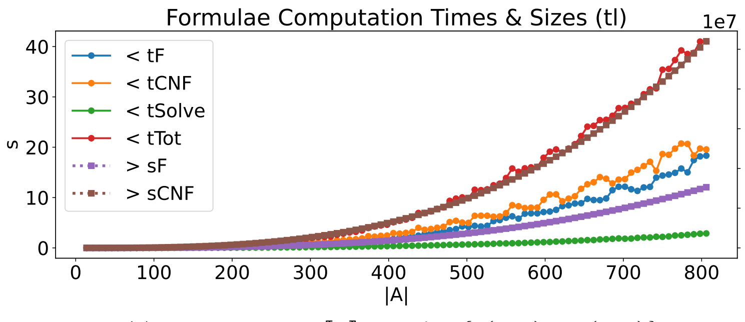
It is easy to prove by induction: for $t \in \mathbb{X}$ and $t \in q$, U^t_q has one element of size 1. For $t = f(t_1, \dots, t_n)$, we say that for all rules $f(q_1, \dots, q_n) \to p \in \Delta$, U^t_q contains the combinations by $\otimes$ of $(p \approx q, \varnothing)$ and of one element of each $U^{t_i}_{q_i}$. The size of each of these combinations is the sum of the size of its parts, that is to say $1 + |t_1| + \cdots + |t_n| = |t|$. The number of these combinations is the product of the sizes $\left|U^{t_i}_{q_i}\right| \leqslant |\Delta|^{|t_i|}$. In total, each rule $f(q_1, \dots, q_n) \to p \in \Delta$ can add no more than $|\Delta|^{|t_1| + \cdots + |t_n|}$ elements. Thus, the total number of elements in U^t_q is no greater than $|\Delta| \times |\Delta|^{|t_1| + \cdots + |t_n|} = |\Delta|^{|t|}$.

From these bounds on U^t_q, we give an upper bound for the size of φ^a (cf. Def. $11_{[p4]}$):

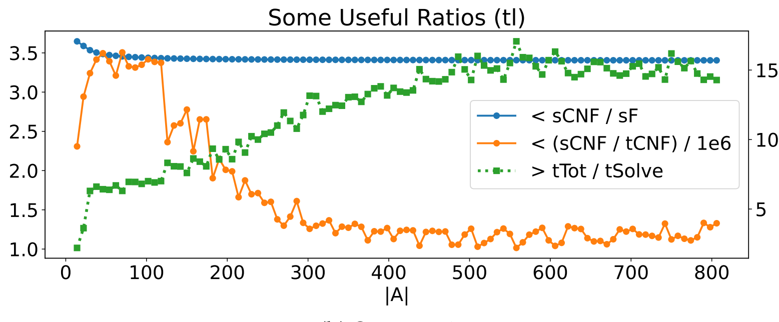
- ⋄ The disjunction on β is of size $O\big(|r| \times |\Delta|^{|r|}\big)$
- ⋄ The conjunction on α, σ is of size $O\big(|l \to r| \times |\Delta|^{|l \to r|}\big)$
- ⋄ The size of the conjunction on q and $l \to r$ is the sum of the above for each rule of $\mathcal{R}$ and state of Q. If let N be the size of the largest rule, we have $O(|Q| \times |\mathcal{R}| \times N \times |\Delta|^N)$. □

To evaluate how SAT solvers deal with those inputs, we conducted various experiments on a proof-of-concept (PoC) implementation of a State-Merging solver [(f)]. Our PoC is implemented in Python 3.11 and uses PySAT [17] to interface with the CNF-SAT–solver backend, Minisat 2.2 [7]. All experiments were run on the same machine: Arch Linux, 12-core AMD Ryzen 9 5900X, 32G RAM, Python 3.11.8. All tests were single-threaded.

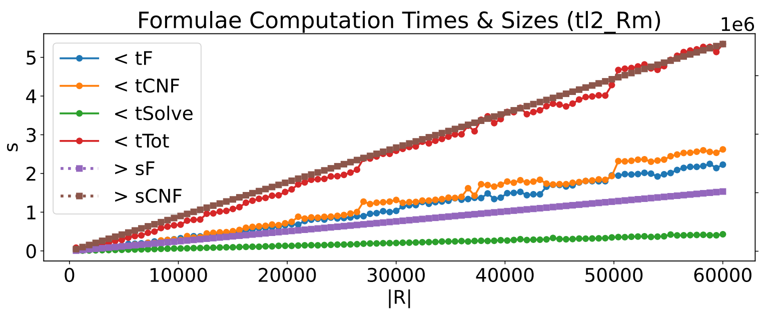
Qualitatively, the PoC does, instantly, produce the right answers on our usual small examples, such as $\mathtt{f}(x) \to \mathtt{f}(\mathtt{a}(\mathtt{a}(x)))$, $\mathtt{f}(x, y) \to \mathtt{f}(\mathtt{a}(x), \mathtt{b}(y))$, etc. Quantitatively, State Merging solving scales quite differently depending on which characteristic of the input is considered, some having a linear impact and some an exponential one, as predicted by Prp. 20. To observe that, we experiment on simple values of $\mathcal{A}, \mathcal{B}, \mathcal{R}$ and graph the solution times and sizes in Figure 1 (mind that some scales are log and some linear).



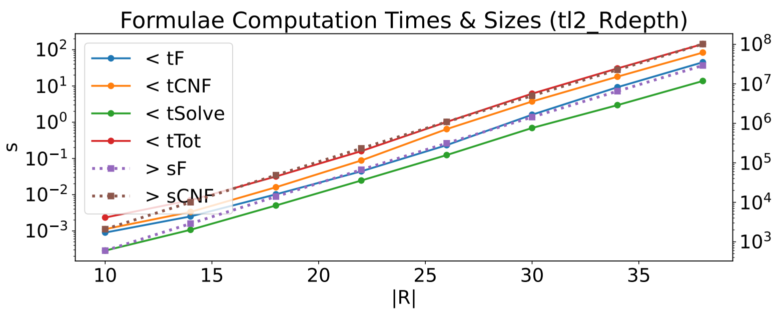
(a) $\mathcal{A} = \mathcal{A}_n$ grows, $[\![\mathcal{B}]\!] = \emptyset$, $\mathcal{R} = \{\, \mathtt{f}(x, y) \to \mathtt{f}(y, x) \,\}$



(b) Same as 1a



(c) $\mathcal{A} = \mathcal{A}_2$, $|\mathcal{A}_2| = 22$, $[\![\mathcal{B}]\!] = \emptyset$, $\mathcal{R} = \mathcal{R}_n = \{\, \mathtt{f}(x_i, y_i) \to \mathtt{f}(y_i, x_i) \mid i \in [\![1, n]\!] \,\}$



(d) $\mathcal{A} = \mathcal{A}_2$, $[\![\mathcal{B}]\!] = \emptyset$, $\mathcal{R} = \mathcal{R}'_n$ grows

Fig. 1: Some experimental results. $<, >$ indicates whether the curve refers to the y-axis on the left or right. `t`=time, `s`=size, `F`=φ, `CNF`=φ in CNF, `tSolve`=time spent in the SAT solver backend, `tTot`=total time to solution.

Our test automaton is in all cases $\mathcal{A}_n$, recognising lists $\mathtt{f}(\mathtt{a}_1, \mathtt{f}(\mathtt{a}_2, \dots \mathtt{f}(\mathtt{a}_k, \bot) \dots)$, $\mathtt{a}_i \in [\![1, n]\!]$, using a separate non-deterministic state for each possible value of leaves $\mathtt{a}_i$. A human being would undoubtedly write that TA in 2 states for any n, but the curves are scarcely affected by the structure of $\mathcal{A}$, so this is as good an example as any. Fig. 1a shows a polynomial growth across all metrics (time and size of φ), as $|\mathcal{A}_n|$ increases. For large formulæ, about $^1/_{15\text{th}}$ of the total time is spent in the SAT-solver, by Fig. 1b (including time spent reading the Python `list-of-list-of-int` structure containing the CNF clauses). The rest is spent computing φ and converting it to Conjunctive Normal Form (CNF; we use the linear Tseytin transformation). The ratios of Fig. 1b tell the same story regardless

[(f)] url: `https://github.com/vincent-hugot/CIAA-2024-SAT-Completion`.

of $A, B, \mathcal{R}$. Our formulæ do not seem to be pathological cases for the solver; we are mostly limited by how fast we can generate φ.

No graph is provided regarding the influence of B because there is not much to discuss: $|\varphi^s|$ is linear in $|B|$, and the tests bear that out. The effect of $\mathcal{R}$ is more complicated. In terms of the number of rules, all metrics grow linearly, viz. Fig. 1c. The size of the rules, however, has an exponential impact. Consider Fig. 1d, where $\mathcal{R}'_n = \{f(0, f(0, \dots f(x, y) \dots)) \to f(\dots f(f(y, x), 0) \dots, 0)\}$ operates on lists of length n – the results are the same with variables $x_1, \dots, x_n$ instead of constants.

The practicality of the approach is therefore highly dependent upon the form of the TRS. In all cases, and even accounting for hardware differences, this is a marked improvement compared to [2], whose own experiments could not exceed $|A| = 20$.

Engineering considerations: A well-engineered implementation could vastly outperform our PoC. Most of the time is spent doing formula manipulation via structural pattern-matching; on a synthetic test on large formulæ, OCaml outperforms Python on such tasks by a factor of 16.4. Furthermore, φ is naturally a large conjunction of independent clauses which could trivially be generated in parallel, fed to the solver incrementally, and discarded. On our 12-core CPU, expecting a 10x time speedup is not unreasonable, and this would also immensely reduce the maximum RAM necessary, decoupling it from the current $O(|\varphi|)$. Those two improvements combined would probably yield a performance increase in the vicinity of 150x on our machine, even if the SAT solver itself is still shared and single-threaded, since it is *very* far from being the bottleneck. Since SAT-solving can itself be parallelised to some extent using divide-and-conquer methods, e.g. [16], massively parallel implementations are theoretically possible.

6 Conclusion and future works

In order to automate the steps of reachability analysis which require expert intervention, we have introduced a SAT encoding of the *state merging problem (SMP)*: **In:** TA A, B, and TRS $\mathcal{R}$; **Out:** $\approx$ such that ${}^{A}/_{\approx}$ is closed wrt. $\mathcal{R}$ and does not intersect B. We have presented various ways to integrate SAT-backed SMP solving into completion algorithms to perform reachability analysis. Our experiments and complexity analysis show that, for relatively flat TRS, the method can be applied to fairly large instances and is potentially quite scalable wrt. available computing power. Its main limitation is the exponential impact of large, deep rewrite rules.

Future works: our main focus is of course to overcome that limitation. An interesting idea involves changing our proxy criterion for "completeness wrt. $\mathcal{R}$" from acriticality to something more permissive, involving partial orderings on states instead of partitioning, such that applying a rewriting rule always leads to a "weaker" state, rather than an equivalent one. As this constraint is strictly easier to satisfy, this would allow not only for our algorithm to find solutions for more instances, but also for solutions to potentially be found with fewer completion steps. The impact of this change is difficult to gauge a priori, but as it loads more work on the highly optimized SAT-solvers and potentially lessens the number of iterations before a solution is reached, we can hope for a performance increase.

From a different angle, any applicable TRS-flattening technique would enable us to blunt the impact of the largest rules.

References

1. Y. Boichut, R. Courbis, P.-C. Héam, and O. Kouchnarenko. Finer is Better: Abstraction Refinement for Rewriting Approximations. In *RTA*, volume 5117 of *LNCS*, pages 48–62. Springer, 2008.
2. Y. Boichut, Th.-B.-H. Dao, and V. Murat. Characterizing conclusive approximations by logical formulae. In *RP 2011, Genoa*, LNCS, 2011.
3. Y. Boichut and P.-C. Héam. A Theoretical Limit for Safety Verification Techniques with Regular Fix-point Computations. *IPL*, 108(1):1–2, 2008.
4. Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Approximation-based tree regular model-checking. *Nord. J. Comput.*, 14(3):216–241, 2008.
5. A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract Regular (Tree) Model Checking. *STTT*, 14(2):167–191, 2012.
6. H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications* (TATA), 2007.
7. Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *SAT 2003*, pages 502–518. Springer, 2003.
8. Guillaume Feuillade, Thomas Genet, and Valérie Viet Triem Tong. Reachability analysis over term rewriting systems. *J. Autom. Reason.*, 33(3-4):341–383, 2004.
9. T. Genet and V. Rusu. Equational Tree Automata Completion. *Journal of Symbolic Computation,*, vol. 45, 2010, 2010.
10. Th. Genet. Decidable Approximations of Sets of Descendants and Sets of Normal Forms. In *RTA*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.
11. Th. Genet. Completeness of tree automata completion. In *FSCD 2018*, volume 108 of *LIPIcs*, pages 16:1–16:20, 2018.
12. Th. Genet, T. Haudebourg, and Th. P. Jensen. Verifying higher-order functions with tree automata. In *FOSSACS 2018*, LNCS, 2018.
13. Th. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *CADE*, volume 1831 of *LNAI*, pages 271–290. Springer-Verlag, 2000.
14. Thomas Genet. *Reachability analysis of rewriting for software verification*. Habilitation thesis (habilitation à diriger des recherches), University of Rennes I, 2009.
15. Thomas Genet and Valérie Viet Triem Tong. Reachability analysis of term rewriting systems with timbuk. In *LPAR 2001*, LNCS, 2001.
16. Marijn JH Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding cdcl sat solvers by lookaheads. In *HVC*, 2011.
17. Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.
18. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich ssertional languages. In *CAV '97*, LNCS, 1997.
19. Nils Klarlund and Anders Møller. *MONA Version 1.4 User Manual*, January 2001. Notes Series NS-01-1. Available from `http://www.brics.dk/mona/`.