Enhancing Approximations for Regular Reachability Analysis

Alois Dreyfus, Pierre-Cyrille Heam, Olga Kouchnarenko

Enhancing Approximations for Regular Reachability Analysis

Aloïs Dreyfus, Pierre-Cyrille Héam, and Olga Kouchnarenko

FEMTO-ST CNRS 6174, University of Franche-Comté & Inria/CASSIS, France
firstname.name@femto-st.fr

Abstract. This paper introduces two mechanisms for computing over-approximations of sets of reachable states, with the aim of ensuring termination of state-space exploration. The first mechanism consists in over-approximating the automata representing reachable sets by merging some of their states with respect to simple syntactic criteria, or a combination of such criteria. The second approximation mechanism consists in manipulating an auxiliary automaton when applying a transducer representing the transition relation to an automaton encoding the initial states. In addition, for the second mechanism we propose a new approach to refine the approximations depending on a property of interest. The proposals are evaluated on examples of mutual exclusion protocols.

1 Introduction and Problem Statement

Reachability analysis is a challenging issue in formal software verification. Since the reachability problem is in general undecidable in most formalisms, several ad-hoc approaches have been developed, such as symbolic reachability analysis using finite representations of infinite sets of states. *Regular model checking* (RMC for short) – a symbolic approach using regular sets to represent sets of states – tackles undecidability in either of two ways: pointing out classes of regulars sets and relations for which the reachability problem is decidable (see for instance [21]), or developing semi-algorithmic and/or approximation-based approaches (see for instance [15,16]) to semi-decide the reachability problem.

In this paper we present new approximation techniques for RMC, with the aim of providing quite efficient (semi-)algorithms. The first technique consists in over-approximating the automata representing reachable sets by merging some of their states with respect to simple syntactic criteria, or a combination of such criteria (Section 2). The second approximation technique consists in using an auxiliary automaton when applying a transducer representing the transition relation to an automaton encoding the initial states (Section 3). Moreover, for the second technique we develop a new approach to refine the approximations, close to the well-known CEGAR technique (Section 4). The proposals are evaluated on examples of mutual exclusion protocols (Section 5). Omitted proofs are available online[1].

[1] http://disc.univ-fcomte.fr/~adreyfus/ciaa13/version_longue.pdf

Related Work. Regular model-checking remains an active research domain in computer science (see [14] and [4] for a thorough overview). In [23] the authors propose to use regular sets of strings to represent states of parametrized arrays of processes, and to represent the effect of performing an action by a predicate transformer (transducer). In this work only transducers representing the effect of a single application of a transition are considered, and consequently the reachability analysis does not terminate for a lot of protocols. To bypass this problem and still reach a fixpoint, the principal methods are acceleration (providing exact computations) [22,11,15,16,3,8], widening (extrapolating) [11,25,24], and automata abstraction [10]. Recently, new results in RMC have been obtained for specific protocols (i.e., CLP [19], communicating systems [20], tree language [1,12], or relational string verification using multi-track automata [26]), using domain-specific techniques [7]. Our contributions aim at improving the generic method in [10] by giving means to build over-approximations by merging abstract states of the system (and not of the transducer, which is never modified). Unlike [11,10], our proposals do not require the subset-construction, minimization and determinization of the obtained automaton at each RMC step.

Formal Background. We assume the reader familiar with basic notions of language theory. An *automaton* $\mathcal{A}$ on an alphabet Σ is a tuple (Q, Σ, E, I, F) where Q is the finite set of *states*, $E \subseteq Q \times \Sigma \times Q$ is the set of *transitions*, $I \subseteq Q$ is the set of *initial states* and $F \subseteq Q$ is the set of *final states*. We define the size of $\mathcal{A}$ by $|\mathcal{A}| = |Q| + |E|$. An automaton is *deterministic* [resp. *complete*] if I is a singleton and for each $(q, a) \in Q \times \Sigma$ there is at most [resp. at least] one $p \in Q$ such that $(q, a, p) \in E$. A path in $\mathcal{A}$ is a (possibly empty) finite sequence of transitions $(p_1, a_1, q_1) \dots (p_n, a_n, q_n)$ such that for each i, $q_i = p_{i+1}$. The integer n is the length of the path and the word $a_1 \dots a_n$ is its label. A path is *successful* if p_1 is initial and p_n is final. A word w is *accepted* by $\mathcal{A}$ if w is the label of a successful path. The set of words accepted by $\mathcal{A}$ is denoted $L(\mathcal{A})$. If $\mathcal{A}$ is deterministic and complete, for every state q and every word w, there exists a unique state of $\mathcal{A}$, denoted $q \cdot_{\mathcal{A}} w$ reachable from q by reading a path labeled by w. If there is no ambiguity on $\mathcal{A}$, it is simply denoted $q \cdot w$. By convention, $q \cdot \varepsilon = \{q\}$. A state q is *accessible* [resp. *co-accessible*] if there exists a path from an initial state to q [resp. if there exists a path from q to a final state]. An automaton whose states are all both accessible and co-accessible is called *trim*. If $\mathcal{A}$ is not a trim automaton, removing from $\mathcal{A}$ all states that are not both accessible and co-accessible together with all related transitions provides an equivalent trim automaton. Let $\mathcal{A}_1 = (Q_1, \Sigma, E_1, I_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, E_2, I_2, F_2)$ be two automata over the same alphabet, the product of $\mathcal{A}_1$ and $\mathcal{A}_2$ is the automaton $(Q_1 \times Q_2, \Sigma, E, I_1 \times I_2, F_1 \times F_2)$, denoted $\mathcal{A}_1 \times \mathcal{A}_2$, where $E = \{((p_1, p_2), a, (q_1, q_2)) \mid (p_1, a, q_1) \in E_1 \wedge (p_2, a, q_2) \in E_2\}$. By definition, $L(\mathcal{A}_1 \times \mathcal{A}_2) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. Let $\hat{\mathcal{A}} = (\hat{Q}, \Sigma, \hat{E}, \hat{I}, \hat{F})$ be the trim automaton obtained from $\mathcal{A}$, given an equivalence relation $\sim \subseteq Q \times Q$, $\mathcal{A}/_\sim$ denotes the automaton $(\hat{Q}/_\sim, \Sigma, E', \hat{I}/_\sim, \hat{F}/_\sim)$ where $E' = \{(\tilde{p}, a, \tilde{q}) \mid \exists p \in \tilde{p}$ and $\exists q \in \tilde{q}$ s.t. $(p, a, q) \in \hat{E}\}$. One can easily check that $L(\mathcal{A}) \subseteq L(\mathcal{A}/_\sim)$. For instance, given the automata of Fig. 1 and the relation $\sim_{\text{exe}}$ whose classes

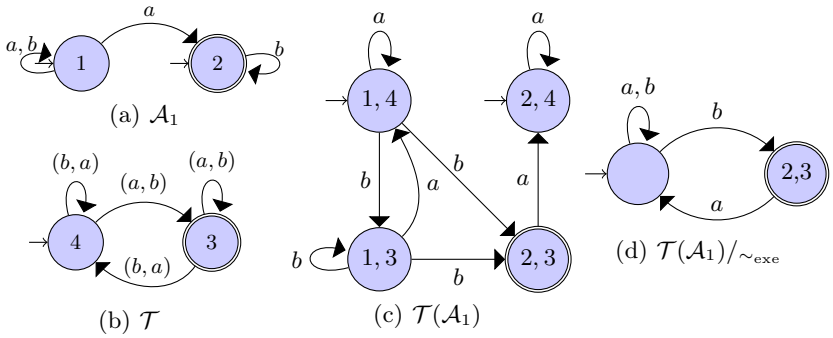


Fig. 1. Illustrating examples

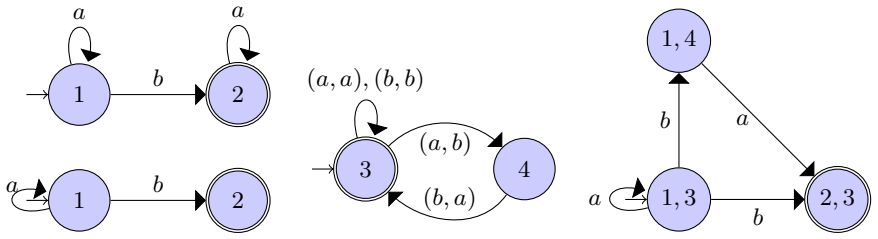
are $\{(1,4),(2,4),(1,3)\}$ and $\{(2,3)\}$, the automaton $\mathcal{T}(\mathcal{A}_1)/_{\sim_{\text{exe}}}$ is depicted on Fig. 1. Two automata $\mathcal{A}_1 = (Q_1, \Sigma, E_1, I_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, E_2, I_2, F_2)$ are isomorphic if there exists a one-to-one function $f : Q_1 \to Q_2$ satisfying $(p, a, q) \in E$ iff $((f(p), a, f(q)) \in E$, and $f(I_1) = I_2$, $f(F_1) = F_2$ when lifted to sets. Informally, two automata are isomorphic if they are equal up to state names.

Let Σ_1 and Σ_2 be two alphabets, a *transducer* on Σ_1, Σ_2 is an automaton on $\Sigma_1 \times \Sigma_2$. Each transducer $\mathcal{T}$ on Σ_1, Σ_2 induces a relation $R_\mathcal{T}$ on $\Sigma_1^* \times \Sigma_2^*$ defined by: for the a_i's in Σ_1 and the b_j's in Σ_2, $(a_1 \dots a_n, b_1 \dots b_m) \in R_\mathcal{T}$ iff $n = m$ and the word $(a_1, b_1) \dots (a_n, b_n)$ is accepted by $\mathcal{T}$. The reflexive transitive closure of $R_\mathcal{T}$ is denoted $R_\mathcal{T}^*$. Let $\mathcal{A} = (Q_1, \Sigma, E_1, I_1, F_1)$ be an automaton on Σ_1, and $\mathcal{T} = (Q_2, \Sigma_1 \times \Sigma_2, E_2, I_2, F_2)$ a transducer on $\Sigma_1 \times \Sigma_2$, we denote by $\mathcal{T}(\mathcal{A})$ the automaton $(Q_1 \times Q_2, \Sigma_2, E, I_1 \times I_2, F_1 \times F_2)$ on Σ_2 where $E = \{((p_1, p_2), b, (q_1, q_2)) \mid (p_1, a, q_1) \in E_1 \wedge (p_2, (a, b), q_2) \in E_2\}$. An example is depicted on Fig. 1. By definition, $L(\mathcal{T}(\mathcal{A}))$ is the set of words v satisfying $(u, v) \in R_\mathcal{T}$ for some words $u \in L(\mathcal{A})$. If $\mathcal{T} = (Q_2, \Sigma_1 \times \Sigma_2, E_2, I_2, F_2)$ is a transducer, we denote by $\mathcal{T}^{-1}$ the transducer $(Q_2, \Sigma_2 \times \Sigma_2, E_2', I_2, F_2)$ with $E_2' = \{(p, (a, b), q) \mid (p, (b, a), q) \in E_2\}$. One can check that $(u, v) \in R_\mathcal{T}$ iff $(v, u) \in R_{\mathcal{T}^{-1}}$.

Regular Reachability Problem. The following regular reachability problem – central for RMC – is known to be undecidable in general; its variants have been addressed in most of the papers in Sect. 1.

Input: Two finite automata $\mathcal{A}$ and $\mathcal{B}$ on a same alphabet Σ, and a transducer $\mathcal{T}$ on $\Sigma \times \Sigma$.
Output: 1 if $R_\mathcal{T}^*(L(\mathcal{A})) \cap L(\mathcal{B}) = \emptyset$, and **0** otherwise.

Since the problem is concerned with the reflexive-transitive closure, we may assume without loss of generality that for every $u \in \Sigma^*$, $(u, u) \in R_\mathcal{T}$. In the rest of the paper, all considered relations contain the identity.



(a) $\mathcal{A}_{\text{tr}}$ (down) and $\mathfrak{Left}[\mathcal{A}_{\text{tr1}}]$ (up)

(b) $\mathcal{T}_{\text{tr}}$

(c) $\mathcal{A}_{\text{tr1}}{=}\mathcal{T}_{\text{tr}}(\mathcal{A}_{\text{tr}})$ after trimming

Fig. 2. Token ring

2 Quotient-based Approximations

This section introduces the first mechanism for computing over-approximations of sets of reachable states, which consists in over-approximating the automata representing reachable sets by merging some of their states. For doing this, basic elementary policies as well as their combinations are introduced.

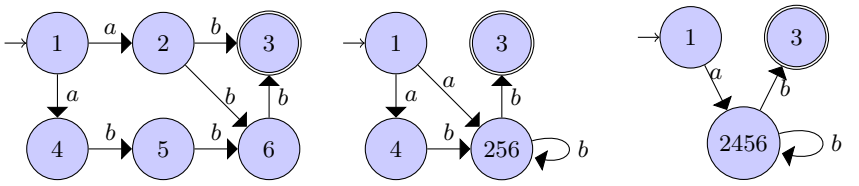
Given an automaton $\mathcal{A}$, we define an *approximation* as a function mapping each automaton $\mathcal{A}$ to an equivalence relation $\sim_{\mathcal{A}}$ over the states of $\mathcal{A}$. The approximation function $\mathfrak{F}$ is *isomorphism-compatible* if for every pair of automata $\mathcal{A}_1$ and $\mathcal{A}_2$, every isomorphism φ from $\mathcal{A}_1$ to $\mathcal{A}_2$, $p \sim_{\mathcal{A}_1} q$ iff $\varphi(p) \sim_{\mathcal{A}_2} \varphi(q)$. We denote $\mathfrak{F}[\mathcal{A}]$ the automaton $\hat{\mathcal{A}}/_{\mathfrak{F}(\hat{\mathcal{A}})}$, where $\hat{\mathcal{A}}$ is the trim automaton obtained from $\mathcal{A}$. We inductively define $\mathfrak{F}^n[\mathcal{A}]$ by $\mathfrak{F}^0[\mathcal{A}] = \mathcal{A}$, and $\mathfrak{F}^n[\mathcal{A}] = \mathfrak{F}[\mathfrak{F}^{n-1}[\mathcal{A}]]$ for $n > 0$.

Let us now introduce two isomorphism-compatible approximation functions. They are easily computable, and represent simple criteria naturally used by the specifier, as for example in [10] for computing equivalence relations, or in [5] for monitoring LTL properties.

- $\mathfrak{Left}$, mapping each automaton (Q, Σ, E, I, F) to the reflexive-transitive closure of the relation R_{left}, defined by $pR_{\text{left}}q$ iff $L(Q, \Sigma, E, I, \{p\}) \cap L(Q, \Sigma, E, I, \{q\}) \neq \emptyset$.
- $\mathfrak{Right}$, mapping each automaton (Q, Σ, E, I, F) to the reflexive-transitive closure of the relation R_{right}, defined by $pR_{\text{right}}q$ iff $L(Q, \Sigma, E, \{p\}, F) \cap L(Q, \Sigma, E, \{q\}, F) \neq \emptyset$.

Let us consider the example of the token ring protocol for which the automata are depicted on Fig. 2. Let $\mathcal{A}_{\text{tr1}}$ be the automaton obtained by trimming $\mathcal{T}_{\text{tr}}(\mathcal{A}_{\text{tr}})$. The relation $\mathfrak{Right}[\mathcal{A}_{\text{tr1}}]$ is the identity relation, therefore $\mathfrak{Right}[\mathcal{A}_{\text{tr1}}] = \mathcal{A}_{\text{tr1}}$. However, for the relation $\mathfrak{Left}$, the states $(1, 4)$ and $(2, 3)$ are equivalent since they can be reached from the initial state by reading b. The automaton $\mathfrak{Left}[\mathcal{A}_{\text{tr1}}]$ is depicted on Fig. 2(a) (up).

Proposition 1. *For each automaton $\mathcal{A}$, if $\mathfrak{F}$ is an isomorphism-compatible approximation function, then the sequence $(\mathfrak{F}^n[\mathcal{A}])_{n\in\mathbb{N}}$ is ultimately constant, up to*



(a) $\mathcal{A}$ (b) $\mathfrak{Right}[\mathcal{A}]$ (c) $\mathfrak{Right}^2[\mathcal{A}]=C_{\mathfrak{Right}}(\mathcal{A})$

Fig. 3. Computing $C_{\mathfrak{Right}}(\mathcal{A})$

Semi-Algorithm `FixPoint`
Input: $\mathcal{A}$, $\mathcal{T}$, $\mathcal{B}$, $\mathfrak{F}$
  If $L(C_{\mathfrak{F}}(\mathcal{T}(\mathcal{A}))) \cap L(\mathcal{B}) \neq \emptyset$ **then**
    return *Inconclusive*
  EndIf
  If $L(C_{\mathfrak{F}}(\mathcal{T}(\mathcal{A}))) = L(\mathcal{A})$ **then**
    return *Safe*
  EndIf
  Return `FixPoint`$(C_{\mathfrak{F}}(\mathcal{T}(\mathcal{A})), \mathcal{T}, \mathcal{B}, \mathfrak{F})$

(a) `FixPoint`

Semi-Algorithm `FixPointT`
Input: $\mathcal{A}$, $\mathcal{T}$, $\mathcal{B}$, $\mathcal{C}$
Variable: k
  k:=0
  While $(L(\mathcal{T}_{\mathcal{C}}^{k+1}(\mathcal{A})) \neq L(\mathcal{T}_{\mathcal{C}}^{k}(\mathcal{A})))$ **do**
    $k := k + 1$
  EndWhile
  If $(L(\mathcal{T}_{\mathcal{C}}^{k}(\mathcal{A})) \cap L(\mathcal{B}) = \emptyset)$ **then**
    Return *Safe*
  Else
    Return *Inconclusive*
  EndIfElse

(b) `FixPointT`

Fig. 4. Fixpoint algorithms

isomorphism. Let $C_{\mathfrak{F}}(\mathcal{A})$ denote the limit of $(\mathfrak{F}^n[\mathcal{A}])_{n\in\mathbb{N}}$. Moreover, if for each automaton $\mathcal{A}$ and each pair of states p, q of $\mathcal{A}$, one can check in polynomial time whether $p \sim_{\mathcal{A}} q$, then $C_{\mathfrak{F}}(\mathcal{A})$ can be computed in polynomial time as well.

In the `FixPoint` algorithm depicted in Fig. 4(a), given a finite automaton $\mathcal{A}$ (state of the system), a transducer $\mathcal{T}$ (transition relation), a finite automaton $\mathcal{B}$ (bad property), and an isomorphism-compatible function $\mathfrak{F}$ (approximation criterion), the first check (emptiness) can be performed in polynomial time. Then, unfortunately, the equality of the languages cannot be checked in polynomial time, since the involved automata are not deterministic. Nevertheless, recently developed algorithms [17,2,9] allow solving this problem very efficiently. Note also that the equality test can be replaced by another test – e.g., isomorphism or (bi)simulation – implying language equality or inclusion, as $L(\mathcal{A}) \subseteq L(C_{\mathfrak{F}}(\mathcal{T}(\mathcal{A}))))$ by construction.

Proposition 2. *The* `FixPoint` *semi-algorithm is correct: if it returns* Safe*, then* $R^*_{\mathcal{T}}(L(\mathcal{A})) \cap L(\mathcal{B}) = \emptyset$.

The approach can be illustrated on the example in Fig. 2 with $\mathfrak{F} = \mathfrak{Left}$: $C_{\mathfrak{Left}}(\mathcal{T}_{\text{tr}}(\mathcal{A}_{\text{tr}})) = \mathfrak{Left}(\mathcal{A}_{\text{tr}})$. One can check that $C_{\mathfrak{Left}}(\mathcal{T}_{\text{tr}}(C_{\mathfrak{Left}}(\mathcal{T}_{\text{tr}}(\mathcal{A}_{\text{tr}}))))$ and

$C_{\mathfrak{Left}}(\mathcal{T}_{\text{tr}}(\mathcal{A}_{\text{tr}}))$ are isomorphic. Therefore `FixPoint` stops after one recursive call and returns *Safe*.

From now on, given two approximation functions $\mathfrak{F}$ and $\mathfrak{G}$, we denote $\mathfrak{F}.\mathfrak{G}$ the approximation function defined by $(\mathfrak{F}.\mathfrak{G})(\mathcal{A}) = \mathfrak{F}(\mathcal{A}) \cap \mathfrak{G}(\mathcal{A})$ for every automaton $\mathcal{A}$. In addition, the approximation function $\mathfrak{F} + \mathfrak{G}$ is defined by: for every automaton $\mathcal{A}$, $(\mathfrak{F} + \mathfrak{G})(\mathcal{A})$ is the smallest equivalence relation containing both $\mathfrak{F}(\mathcal{A})$ and $\mathfrak{G}(\mathcal{A})$. Then using several approximation functions and combining them allow us to obtain new – stronger or weaker – approximations. Section 5 gives experimental results for the $\mathfrak{Left}$, $\mathfrak{Right}$ approximations together with the $\mathfrak{In}$ and $\mathfrak{Out}$ approximations, and for their combinations.

3 Transducer-based Approximations

This section introduces another approximation mechanism consisting in reasoning about the application of k copies of a transducer representing the transition relation to an automaton representing the initial states. The states reached in the transducers are encoded as a finite word, and an additional automaton is used for specifying what are the combinations of transducer states that have to be merged. This technique is inspired by an automata theoretic construction in [11], with the difference concerning the equivalence relation, and the use of automata at step k (the transducer is never modified).

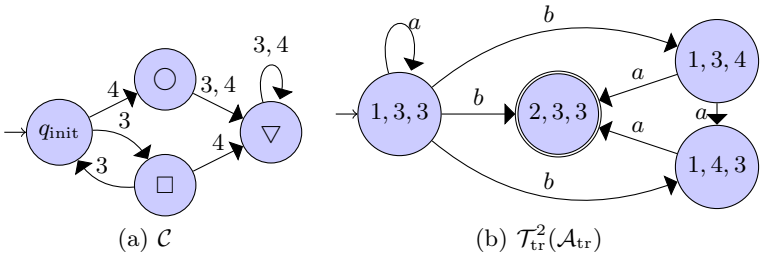
Let $\mathcal{A} = (Q, \Sigma, E, I, F)$ be a finite automaton, $\mathcal{T} = (Q_T, \Sigma \times \Sigma, E_T, I_T, F_T)$ a transducer, and $\mathcal{C} = (Q_C, Q_T, E_C, \{q_{\text{init}}\}, \emptyset)$ a deterministic complete finite automaton on Q_T (i.e., the transitions of $\mathcal{C}$ are labeled with states of $\mathcal{T}$). Let φ_k be a one-to-one mapping from the set $(((Q \times Q_T) \times Q_T) \dots \times Q_T)$ of states of $\mathcal{T}^k(\mathcal{A})$ to $Q \times Q_T^k$, where Q_T^k is the set of words of length k on Q_T. We set a relation $\sim_{\mathcal{C}}$ on states of $\mathcal{T}^k(\mathcal{A})$ as follows: if p and q are states of $\mathcal{T}^k(\mathcal{A})$ such that $\varphi_k(p) = (p_0, w_p)$ and $\varphi_k(q) = (q_0, w_q)$, then $p \sim_{\mathcal{C}} q$ iff $p_0 = q_0$ and $q_{\text{init}} \cdot w_p = q_{\text{init}} \cdot w_q$. The automaton $\mathcal{T}^k(\mathcal{A})/_{\sim_{\mathcal{C}}}$ is denoted $\mathcal{T}^k_{\mathcal{C}}(\mathcal{A})$. One can easily check that $\sim_{\mathcal{C}}$ is an equivalence relation.

Let us consider again $\mathcal{A}_{\text{tr}}$ and $\mathcal{T}_{\text{tr}}$ from Fig. 2. We consider the automaton $\mathcal{C}$ depicted in Fig. 5(a). The automaton $\mathcal{T}^2_{\text{tr}}(\mathcal{A}_{\text{tr}})$ (after trimming) is depicted in Fig. 5(b). The automata $\mathcal{T}_{\text{tr}}(\mathcal{A}_{\text{tr}})/_{\sim_{\mathcal{C}}}$ and $\mathcal{T}^2_{\text{tr}}(\mathcal{A}_{\text{tr}})/_{\sim_{\mathcal{C}}}$ are depicted in Fig. 6. For instance, in $\mathcal{T}^2_{\text{tr}}(\mathcal{A}_{\text{tr}})$ states $(1, 3, 4)$ and $(1, 4, 3)$ are $\sim_{\mathcal{C}}$-equivalent since they have both 1 as the first element, and $q_{\text{init}} \cdot 34 = q_{\text{init}} \cdot 43 = \bigtriangledown$.

Proposition 3. *An automaton isomorphic to $\mathcal{T}^k(\mathcal{A})/_{\sim_{\mathcal{C}}}$ can be computed in polynomial time in k and in the sizes of $\mathcal{A}$, $\mathcal{T}$ and $\mathcal{C}$.*

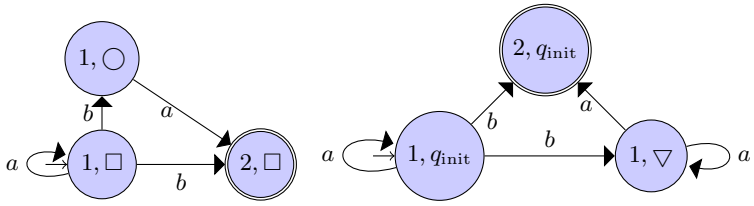
Now, given a finite automaton $\mathcal{B}$, we can use the computed automata when applying the `FixPointT` semi-algorithm described in Fig. 4(b). It may provide an over-approximation of reachable states: if `FixPointT` stops on a not too coarse approximation we can deduce that $\mathcal{R}^*_{\mathcal{T}}(L(\mathcal{A})) \cap L(\mathcal{B}) = \emptyset$. The proof of Proposition 4 is similar to this of Proposition 2.

Proposition 4. *The* `FixPointT` *semi-algorithm is correct: if it returns safe then $\mathcal{R}^*_{\mathcal{T}}(L(\mathcal{A})) \cap L(\mathcal{B}) = \emptyset$.*



(a) $\mathcal{C}$ (b) $\mathcal{T}^2_{\text{tr}}(\mathcal{A}_{\text{tr}})$

Fig. 5. Token ring: Transducer-based approximation (1)



(a) $\mathcal{T}_{\text{tr}}(\mathcal{A}_{\text{tr}})/_{\sim_{\mathcal{C}}}$ (b) $\mathcal{T}^2_{\text{tr}}(\mathcal{A}_{\text{tr}})/_{\sim_C}$

Fig. 6. Token ring: Transducer-based approximation (2)

4 Refining Transducer-based Approximations

In this section we propose to refine transducer-based approximations when the approximate iteration is inconclusive. Intuitively, this happens when the sequence of approximations is too coarse: the result intersects with the set of bad states after k steps while the backward iteration of k copies of the transducer from the bad states does not intersect with the initial states. Our algorithm can be seen as a kind of CEGAR algorithms – the paradigm introduced in [13] and intensively studied during the last decade (see for example [10,6]), with the aim of obtaining finer approximations/abstractions by exploiting counter-examples.

Proposition 5. *If* $L(\mathcal{T}^k_{\mathcal{C}}(\mathcal{A})) \cap L(\mathcal{B}) \neq \emptyset$*, then either* $L(\mathcal{A}) \cap L(\mathcal{T}^{-k}(\mathcal{B})) \neq \emptyset$*, or there exists* j*,* $0 \leq j \leq k$ *such that* $L(\mathcal{T}^j_{\mathcal{C}}(\mathcal{A})) \cap L(\mathcal{T}^{j-k}(\mathcal{B})) \neq \emptyset$ *and* $L(\mathcal{T}(\mathcal{T}^{j-1}_{\mathcal{C}}(\mathcal{A}))) \cap L(\mathcal{T}^{j-k}(\mathcal{B})) = \emptyset$*.*

Assume that $L(\mathcal{T}^j_{\mathcal{C}}(\mathcal{A})) \cap L(\mathcal{T}^{j-k}(\mathcal{B})) \neq \emptyset$ and $L(\mathcal{T}(\mathcal{T}^{j-1}_{\mathcal{C}}(\mathcal{A}))) \cap L(\mathcal{T}^{j-k}(\mathcal{B})) = \emptyset$. As it is classically done in the CEGAR framework, one can compute a relation $\equiv$ on $\mathcal{T}^j_{\mathcal{C}}(\mathcal{A})$ such that $\equiv \subseteq \sim_{\mathcal{C}}$ and $L(\mathcal{T}^j_{\mathcal{C}}(\mathcal{A}))/_{\equiv} \cap L(\mathcal{T}^{k-j}(\mathcal{B})) = \emptyset$. The existence of $\equiv$ is trivial since the results hold for the identity relation. However, when using the CEGAR approach, our goal is to compute a relation $\equiv$ as large as possible, with the aim of ensuring termination of state-space exploration.

To achieve this goal, several heuristics may be used. Instead of computing the $\equiv$ relation, building the corresponding $\mathcal{T}^j_{\mathcal{C}}(\mathcal{A})/_{\equiv}$ automaton, and then performing the fixpoint computation, we propose to use a dynamic approach. More

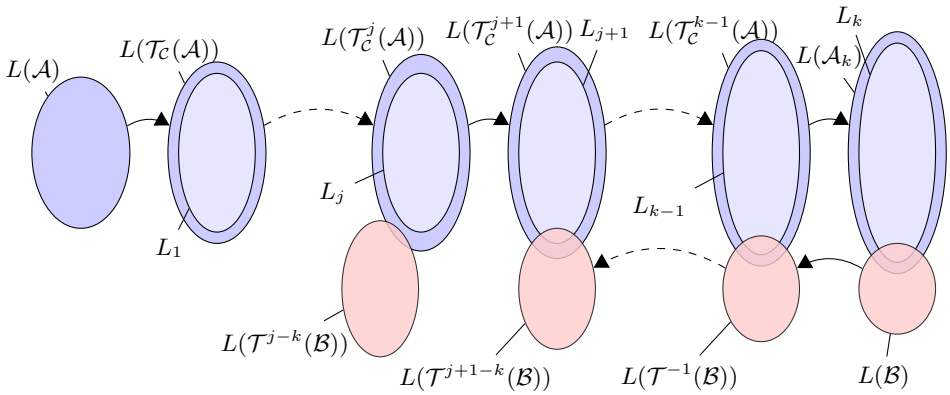


Fig. 7. Refinement: L_i's represent the languages $L(\mathcal{T}(\mathcal{T}_\mathcal{C}^{i-1}(\mathcal{A}))$'s.

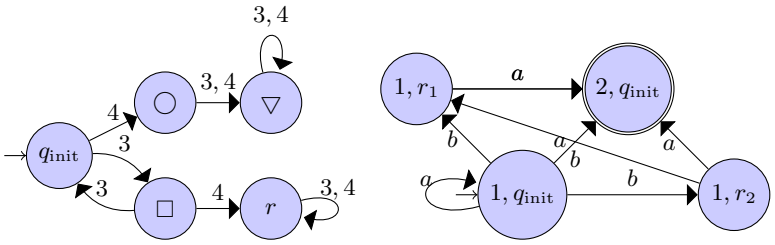
Algorithm `Split`
Input: $S = (Q_S, Q_T, E_S, \{q_0\}, \emptyset)$ a deterministic automaton, $p, q \in Q_S$ and $\alpha, \beta \in Q_T$ such that $p \cdot_S \alpha = q \cdot_S \beta$
$Q'_S := Q_S \cup \{r\}$ where $r \notin Q_S$
$I'_S := \{q_0\}$
$E'_S := E_S \setminus \{(q, \beta, q \cdot_S \beta)\}$
$E'_S := E'_S \cup \{(q, \beta, r)\} \cup \{(r, a, s) \mid (p \cdot \alpha, a, s) \in E_S \text{ and } s \in Q_S \setminus \{p \cdot_S \alpha\}\}$
$E'_S := E'_S \cup \{(r, a, r) \mid (p \cdot \alpha, a, p \cdot \alpha) \in E_S\}$
Return $(Q'_S, Q_T, E'_S, I'_S, \emptyset)$

Fig. 8. Algorithm `Split`

precisely, we prefer to modify $\mathcal{C}$ according to $\equiv$ to avoid similar states merging which may lead to a coarser over-approximation. To modify $\mathcal{C}$ according to $\equiv$, we propose to use the algorithms in Figs. 8 and 10. The `Split` algorithm modifies the given deterministic automaton to provide a weaker abstraction. Its idea is quite natural: if two equivalent states must be distinguished, the automaton $\mathcal{C}$ is refined to take this constraint into account. For example, Figure 9(a) displays the automaton $\mathcal{C}'$ resulting from $\texttt{Split}(\mathcal{C}, \bigcirc, \square, 3, 4)$, where $\mathcal{C}$ is the automaton from Fig. 5(a). The `Split` algorithm dissociating two states, can be used so far as necessary to obtain the refined approximation in the `Refine` algorithm in Fig. 10.

Proposition 6. *The* `Refine` *algorithm always terminates.*

For example, let us consider the $\equiv$ relation whose classes are $\{1, \square, 4\}, \{(2, \square, 3)\}$, $\{(1, \bigcirc, 3), (1, \bigcirc, 4)\}$ and $\{(1, \square, 3)\}$. We apply the `Refine` algorithm to the automata $\mathcal{T}_{\text{tr}}$ (Fig. 2(b)), $\mathcal{C}$ (Fig. 5(a)), $\mathcal{T}_{\text{tr}}(\mathcal{A}_{\text{tr}})/_{\sim_\mathcal{C}}$ (Fig. 6(a)). Since $(1, \bigcirc, 3) \sim_\mathcal{C} (1, \square, 4)$, $\sim_\mathcal{C} \not\subseteq \equiv$. Therefore, the algorithm may compute $\mathcal{C}' = \texttt{Split}(\mathcal{C}, \bigcirc, \square, 3, 4)$ as depicted in Fig. 9(a). Then one can check that $\sim_{\mathcal{C}'} \subseteq \equiv$. The automaton $\mathcal{T}_{\text{tr}}(\mathcal{T}_{\text{tr}}(\mathcal{A}_{\text{tr}})/_{\sim_\mathcal{C}})/_{\sim_{\mathcal{C}'}}$ is depicted in Fig. 9(b).



(a) $\mathcal{C}' = \texttt{Split}(\mathcal{C}, \bigcirc, \square, 3, 4)$ (b) $\mathcal{T}_{\text{tr}}(\mathcal{T}_{\text{tr}}(\mathcal{A}_{\text{tr}})/_{\sim_{\mathcal{C}}})/_{\sim_{\mathcal{C}'}}$

Fig. 9. Examples for the `Split` and `Refine` algorithms

Algorithm `Refine`
Input: $\mathcal{T}$ (transducer), $\mathcal{C}$ a deterministic automaton, $S = (Q_S \times Q_C, Q, E, \{q_0\}, F_S)$ a finite automaton, a relation $\equiv$ such that $\equiv \subseteq \sim_{\mathcal{C}}$ and $L(\mathcal{T}_{\mathcal{C}}(\mathcal{A}))/_{\equiv} \cap L(\mathcal{T}^{-1}(\mathcal{B})) = \emptyset$
While $(\sim_{\mathcal{C}} \not\subseteq \equiv)$ **do**
Choose (p, q, α) **and** (p, q', α') states of $\mathcal{T}(S)$ **such that**
$(p, q, \alpha) \sim_{\mathcal{C}} (p, q', \alpha')$ but $(p, q, \alpha) \not\equiv (p, q', \alpha')$
$\mathcal{C} :=$`Split`$(\mathcal{C}, q, \alpha, q', \alpha')$
EndWhile
Return $\mathcal{C}$

Fig. 10. Algorithm `Refine`

If $L(\mathcal{T}_{\mathcal{C}}^{k}(\mathcal{A})) \cap L(\mathcal{B}) \neq \emptyset$ and $L(\mathcal{A}) \cap L(\mathcal{T}^{-k}(\mathcal{B})) = \emptyset$, then we denote by $\mathtt{J}(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{T}, k)$ the maximal integer j such that $0 \leq j \leq k$ and $L(\mathcal{T}_{\mathcal{C}}^{j}(\mathcal{A})) \cap L(\mathcal{T}^{j-k}(\mathcal{B})) \neq \emptyset$ and $L(\mathcal{T}(\mathcal{T}_{\mathcal{C}}^{j-1}(\mathcal{A}))) \cap L(\mathcal{T}^{j-k}(\mathcal{B})) = \emptyset$. Now, the `Reach-CEGAR` semi-algorithm in Fig. 11 encodes the whole approach: each time a too strong approximation is detected, it is refined. This semi-algorithm may terminate by returning *Safe* if an over-approximation of accessible states that does not contain any bad states. It may also terminate by returning *Unsafe* if it detects a reachable bad state. It may also diverge if the computed approximations have to be refined again and again.

5 Experimental Results

Thanks to a prototype tool, the present paper's proposals have been evaluated on the well-known examples of the Bakery algorithm by Lamport, the token ring algorithm, Dijkstra's, and Burns [25] protocols.

For the quotient-based approximations (Sect. 2), the results are displayed in Fig. 13. In addition to $\mathfrak{Left}$ and $\mathfrak{Right}$, two additional simple isomorphism-compatible approximations are examined:
– $\mathfrak{In}$, mapping each automaton (Q, Σ, E, I, F) to the reflexive-transitive closure of the relation R_{in}, defined by $pR_{\text{in}}q$ iff $\{a_p \in \Sigma \mid \exists p' \in Q, (p', a_p, p) \in E\} = \{a_q \in \Sigma \mid \exists q' \in Q, (q', a_q, q) \in E\}$; and
– $\mathfrak{Out}$, mapping each automaton (Q, Σ, E, I, F) to the reflexive-transitive closure

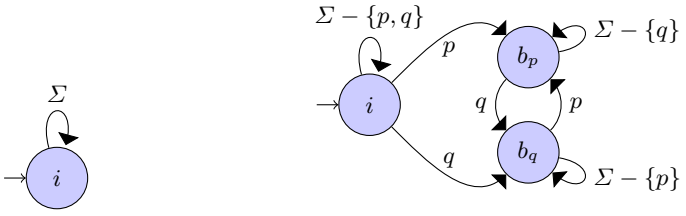
Semi-Algorithm `Reach-CEGAR`
Input: $\mathcal{A}, \mathcal{B}$ finite automata, $\mathcal{T}$ (transducer), $\mathcal{C}$ a deterministic automaton, an integer ℓ
Variables: integers j, k, and equivalence relation $\equiv$

$k := \ell$
While $(L(\mathcal{T}_{\mathcal{C}}^{k}(\mathcal{A})) \cap L(\mathcal{B}) = \emptyset$ **and** $L(\mathcal{T}_{\mathcal{C}}^{k+1}(\mathcal{A})) \neq L(\mathcal{T}_{\mathcal{C}}^{k}(\mathcal{A}))$) **do**
  $k := k + 1$
EndWhile
If $(L(\mathcal{T}_{\mathcal{C}}^{k+1}(\mathcal{A})) = L(\mathcal{T}_{\mathcal{C}}^{k}(\mathcal{A}))$ **and** $L(\mathcal{T}_{\mathcal{C}}^{k}(\mathcal{A})) \cap L(\mathcal{B}) = \emptyset$) **then**
  Return *Safe*
EndIf
If $L(\mathcal{A}) \cap L(\mathcal{T}^{-k}(\mathcal{B})) \neq \emptyset$ **then**
  Return *Unsafe*
EndIf
$j := \mathtt{J}(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{T}, k)$
Let $\equiv$ be such that $\equiv \subseteq \sim_{\mathcal{C}}$ and $L(\mathcal{T}_{\mathcal{C}}^{j}(\mathcal{A}))/_{\equiv} \cap L(\mathcal{T}^{k-j}(\mathcal{B})) = \emptyset$
Return `Reach-CEGAR`$(\mathcal{A}, \mathcal{T}^{-k}(\mathcal{B}), \mathcal{T}, \mathtt{Refine}(\mathcal{T}, \mathcal{C}, \mathcal{T}^{j}(\mathcal{A}), \equiv), j)$

Fig. 11. Semi-algorithm `Reach-CEGAR`

of the relation R_{out}, defined by $pR_{\text{out}}q$ iff $\{a_p \in \Sigma \mid \exists p' \in Q, (p, a_p, p') \in E\} = \{a_q \in \Sigma \mid \exists q' \in Q, (q, a_q, q') \in E\}$.
In Fig. 13, the first column describes the protocol to verify: its name, the size (i.e., $|Q| + |E|$) of the initial automaton, and that of the transducer. The remaining columns give the results for each specific criterion: the first line gives the step of the language equality, or No when not reached; the second line indicates the step when the intersection with the bad-property language is non empty, or $\emptyset$ if it remains empty; the third line gives the size of the last obtained automaton. If a step of the languages equality occurs while having the empty intersection with the bad-property language (cf. values highlighted in bold), the protocol is safe.

For the refinement method, the above mentioned protocols have been studied using different kinds of $\mathcal{C}$-automata: either a *one-state* $\mathcal{C}$, or a *specific* $\mathcal{C}$. When starting the refinement with a one-state $\mathcal{C}$ in Fig. 12(a), all the states are obviously considered as $\mathcal{C}$-equivalent. On the contrary, a specific $\mathcal{C}$ models a property of interest. For example, if two consecutive a are forbidden, and there is a transition $(p, (x, a), q)$ in the transducer of the considered protocol, then the specific $\mathcal{C}$ is like in Fig. 12(b). The two token ring protocols are shown to be safe in four steps using the refinement approach with a one-state automaton. Dijkstra's protocol was proved safe without refinement in 15 steps using a specific automaton. The Bakery and Burns protocols are proved safe in respectively 6 and 14 steps, by using the refinement and specific automata. For all these protocols, the obtained automata have sizes similar to the sizes of the input automata: there is no state explosion. To conclude, the experiments show that our techniques work for all the considered cases, and that they are complementary.



(a) one-state $\mathcal{C}$

(b) specific $\mathcal{C}$ for mutual exclusion protocols

Fig. 12. Different kinds of $\mathcal{C}$ automata

	$\mathfrak{In}$	$\mathfrak{Out}$	$\mathfrak{In}+\mathfrak{Out}$	$\mathfrak{In}.\mathfrak{Out}$	$\mathfrak{Left}$	$\mathfrak{Right}$	$\mathfrak{L}+\mathfrak{R}$	$\mathfrak{L}.\mathfrak{R}$	$(\mathfrak{L}+\mathfrak{R}).(\mathfrak{In}+\mathfrak{Out})$
Token ring	**Step 3**	**Step 3**	**Step 3**	**Step 4**	**Step 3**	**Step 2**	**Step 2**	**Step 3**	**Step 3**
size I : 4	∅	∅	∅	∅	∅	∅	∅	∅	∅
size T : 6	8	8	5	12	8	5	5	8	5
Token ring	**Step 3**	**Step 3**	**Step 3**	**Step 4**	T.o(Step 10)	**Step 2**	**Step 2**	T.o(Step 10)	**Step 3**
size I : 4	∅	∅	∅	∅	∅	∅	∅	∅	∅
size T : 9	8	8	5	12	109	5	5	109	5
Dijkstra	**Step 6**	**Step 6**	**Step 5**	**Step 7**	T.o(Step 10)	**Step 5**	**Step 5**	T.o(115 hours)	**Step 5**
size I : 5	∅	∅	∅	∅	∅	∅	∅	T.o(115 hours)	∅
size T : 62	49	118	20	246	745	11	10	T.o(115 hours)	15
Bakery	**Step 7**	No	No	**Step 10**	T.o(Step 10)	No	No	T.o(Step 10)	No
size I : 2	∅	Step 7	Step 6	∅	∅	Step 3	Step 3	∅	Step 6
size T : 24	43	75	89	97	368	31	31	1253	101
Burns	No	**Step 6**	No	No	No	No	No	No	No
size I : 2	Step 5	∅	Step 3	Step 7	Step 4	Step 3	Step 3	Step 4	Step 3
size T : 22	100	46	53	365	22	18	18	50	53

Fig. 13. Results with syntactic criteria

6 Conclusion

Developing efficient approximation-based techniques is a critical challenging issue to tackle reachability problems when exact approaches do not work. In this paper two new approximation techniques for the regular reachability problem have been presented. Our techniques use polynomial time algorithms, provided that recent algorithms for checking automata equivalence are used; the only exception being language inclusion testing as in [17,2,9]. As a future direction, we plan to upgrade our refinement approach, both on the precision of the approximations and on computation time. Another possible direction is to generalize our approximation mechanisms and to apply them to other RMC applications, e.g., counter systems or push-down systems.

References

1. P. Abdulla, B. Jonsson, P. Mahata, and J. d'Orso. Regular tree model checking. In *CAV*, page 452–466, 2002.
2. P.A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In Esparza and Majumdar [18], pages 158–174.

3. P.A. Abdulla, B. Jonsson, M. Nilsson, and J. d'Orso. Algorithmic improvements in regular model checking. In *CAV*, pages 236–248. Springer, 2003.
4. C. Baier, J.P. Katoen, and Inc ebrary. *Principles of model checking*, volume 950. MIT press, 2008.
5. A. Bauer and Y. Falcone. Decentralised LTL monitoring. In D. Giannakopoulou and D. Méry, editors, *FM*, volume 7436 of *LNCS*, pages 85–100. Springer, 2012.
6. Y. Boichut, R. Courbis, P.-C. Héam, and O. Kouchnarenko. Finer is better: Abstraction refinement for rewriting approximations. In A. Voronkov, editor, *RTA*, volume 5117 of *LNCS*, pages 48–62. Springer, 2008.
7. B. Boigelot. Domain-specific regular acceleration. *STTT*, 14(2):193–206, 2012.
8. B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large. In *CAV*, page 223–235, 2003.
9. F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. Technical report, January 2012. 13p.
10. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *CAV*, page 378–379, 2004.
11. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV*, page 403–418, 2000.
12. A. Bouajjani and T. Touili. Widening techniques for regular tree model checking. *STTT*, page 1–21, 2011.
13. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.
14. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking. 2000*. MIT press, 2000.
15. D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In *CAV*, page 286–297, 2001.
16. D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. *Journal of Logic and Algebraic Programming*, 52:109–127, 2002.
17. L. Doyen and J.-F. Raskin. Antichain algorithms for finite automata. In Esparza and Majumdar [18], pages 2–22.
18. J. Esparza and R. Majumdar, editors. *TACAS*, volume 6015 of *LNCS*. Springer, 2010.
19. F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni. Program specialization for verifying infinite state systems: An experimental evaluation. *Logic-Based Program Synthesis and Transformation*, page 164–183, 2011.
20. T. Le Gall and B. Jeannet. Lattice automata: A representation for languages on infinite alphabets, and some applications to verification. In *SAS*, volume 4634 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2007.
21. A. Cano Gómez, G. Guaiana, and J.-E. Pin. When does partial commutative closure preserve regularity? In *ICALP (2)*, volume 5126 of *LNCS*, pages 209–220. Springer, 2008.
22. B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. *TACAS*, page 220–235, 2000.
23. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *CAV*, page 424–435, 1997.
24. A. Legay. Extrapolating (omega-) regular model checking. *STTT*, 14(2):119–143, 2012.
25. T. Touili. Regular model-checking using widening techniques. In *VEPAS*, volume 50 of *ENTCS*, pages 342–356, 2001.
26. F. Yu, T. Bultan, and O. Ibarra. Relational string verification using multi-track automata. *IJFCS*, 22:290–299, 2011.