INFINITY 2004 Preliminary Version

Regular Model Checking Using Inference of Regular Languages

Peter Habermehl [1,2]

LIAFA, University Paris 7, Case 7014, 2, place Jussieu, F-75251 Paris Cedex 05, France

Tomáš Vojnar [3,4]

FIT, Brno University of Technology, Božetěchova 2, CZ-61266, Brno, Czech Republic

Abstract

Regular model checking is a method for verifying infinite-state systems based on coding their configurations as words over a finite alphabet, sets of configurations as finite automata, and transitions as finite transducers. We introduce a new general approach to regular model checking based on *inference of regular languages*. The method builds upon the observation that for infinite-state systems whose behaviour can be modelled using length-preserving transducers, there is a finite computation for obtaining *all* reachable configurations up to a certain length n. These configurations are a (positive) sample of the reachable configurations of the given system, whereas all other words up to length n are a negative sample. Then, methods of inference of regular languages can be used to generalize the sample to the full reachability set (or an overapproximation of it). We have implemented our method in a prototype tool which shows that our approach is competitive on a number of concrete examples. Furthermore, in contrast to all other existing regular model checking methods, *termination is guaranteed* in general for *all* systems with regular sets of reachable configurations. The method can be applied in a similar way to dealing with reachability relations instead of reachability sets too.

Key words: regular model checking, inference of regular languages

1 Introduction

Regular model checking [13,6] is a promising approach for automatic verification of infinite-state systems. The key advantages of regular model checking are its high degree of automation and the fact that many different kinds of systems—such as pushdown systems, (lossy) FIFO-channel systems, systems with counters, or parameterised and dynamic networks of processes—can be modelled and verified in

[1] Supported by the French ministry of research (ACI project Securité Informatique).
[2] Email: Peter.Habermehl@liafa.jussieu.fr
[3] Supported by the Czech Grant Agency (projects GA CR 102/04/0780 and GA CR 102/03/D211).
[4] Email: vojnar@fit.vutbr.cz

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

a uniform way. Regular model checking is based on the idea of encoding configurations of systems as words over a finite alphabet and transitions as finite-state transducers mapping configurations to configurations. Finite automata can then be naturally used to represent and manipulate (potentially infinite) sets of configurations, and reachability analysis can be performed by computing transitive closures of transducers [12,8,3,4] or images of automata by iteration of transducers [6,18]—depending on whether dealing with *reachability relations* or *reachability sets* is preferred. Correctness of the system being verified may then be checked by comparing the computed reachability set or reachability relation with a set or relation describing the undesirable states or changes of states of the system (encoded again by a finite automaton or transducer).

To facilitate *termination of regular model checking*, which is in general not guaranteed as the problem being solved is undecidable, various *acceleration methods* have been proposed. They include, e.g., widening [6,18], collapsing of automata states based on the history of their creation by composing transducers [12,3], or abstraction of automata [5]. These methods allow regular model checking to terminate on many practical examples and sometimes ensure completeness for subclasses of the systems that may be modelled in the given framework (as, e.g., systems with the so-called bounded local depth or simple transition relations [12]).

In this paper, we introduce a new general method of regular model checking based on using *inference of regular languages* extending the work of [11] (see related work below). The approach is motivated by the observation that for infinite-state systems whose behaviour can be modelled using length-preserving transducers, there is a finite computation for obtaining all reachable configurations up to a certain length. These configurations may be considered as a sample of the reachable configurations of the given system. Then, methods that have been developed for inference of regular languages (we, in particular, build upon the Trakhtenbrot-Barzdin algorithm [19]) may be used to generalize the sample with the aim of obtaining the full reachability set or an overapproximation of it that is precise enough to prove the property of interest (if it holds). As shown by our experiments, the method provides us with *good performance results*. At the same time, in contrast to all other existing regular model checking methods, *termination is guaranteed* for *all* the systems whose set of reachable configurations is regular (including, e.g., lossy channel systems and push-down systems). Similar results can then be obtained for dealing with reachability relations instead of reachability sets too.

From the above, it is clear that we primarily concentrate on systems that may be encoded using length-preserving transducers. Dealing with length-preserving transducers is, however, sufficient even for verification of safety properties of non-length preserving systems. In such a case, words encoding configurations may be in advance extended with special blank symbols to be consumed whenever new useful symbols are to be inserted. Moreover, as we show, our method can be extended to deal with non-length-preserving transducers too. Then, however, our completeness results do not hold (though in practice, the method still behaves well).

We have implemented the method and tested it on a number of examples of different systems including parametric networks of processes, a pushdown system, systems with counters, a system with lossy queues, and a system with a linked list as a representative of systems with recursive data structures. The experiments show that our method is very efficient with the results being mostly comparable with those of [5], which is among the fastest methods known in the field but for which (unlike for our method) no completeness results are known as yet.

Related work. The idea of using inference of regular languages for regular model checking has already been used in [11], which, however, primarily targeted parameterized rings only, and the computation loop was different and required a certain manual classification of the transitions of the systems used. Furthermore, the authors of [11] have not implemented their method. Very recently there appeared [21] another work on verification based on inference of regular languages that is specialised to verification of safety of systems with FIFO channels.

Plan of the paper. We first introduce basic concepts from the area of automata theory and regular model checking. Then, we describe the Trakhtenbrot-Barzdin algorithm for inference of regular languages and propose a way to use it for regular model checking. Finally, we discuss the experiments we have done and conclude.

2 Automata, Transducers and Regular Model Checking Basics

A *finite automaton* is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q$ a set of transitions, $q_0 \in Q$ an initial state and $F \subseteq Q$ a set of final states. M is *deterministic* if $\forall q \in Q, a \in \Sigma$ there exists at most one q' with $(q, a, q') \in \delta$. The transition relation $\rightarrow \subseteq Q \times \Sigma^* \times Q$ of M is defined as the smallest relation satisfying: (1) $\forall q \in Q : q \xrightarrow{\varepsilon} q$, (2) if $(q, a, q') \in \delta$, then $q \xrightarrow{a} q'$, and (3) if $q \xrightarrow{w} q'$ and $q' \xrightarrow{a} q''$, then $q \xrightarrow{wa} q''$. The language recognized by M from a state $q \in Q$ is defined by $L(M, q) = \{w \mid \exists q' \in F : q \xrightarrow{w} q'\}$. The language $L(M)$ is equal to $L(M, q_0)$. A set $L \subseteq \Sigma^*$ is *regular* iff there exists a finite automaton M such that $L = L(M)$. We also define the languages of words up to a certain length: $L^{\leq n} = \{w \in L \mid |w| \leq n\}$, $L^{\leq n}(M, q) = \{w \in L(M, q) \mid |w| \leq n\}$, and $\Sigma^{\leq n} = \{w \in \Sigma^* \mid |w| \leq n\}$. We define the *depth of an automaton* $M = (Q, \Sigma, \delta, q_0, F)$ denoted d_M as the maximum length of the shortest paths leading to the particular states of M from the initial state, i.e. $d_M = max_{q \in Q}\ min_{w \in \Sigma^* \wedge q_0 \xrightarrow{w} q} |w|$. We say that two states $q, q' \in Q$ of M are *k-indistinguishable*, which we denote by $q \equiv_k q'$, if $L^{\leq k}(M, q) = L^{\leq k}(M, q')$. We then define the *degree of distinguishability* ρ_M of M as the minimal k such that any two states q, q' of M are k-distinguishable, i.e. $q \not\equiv_k q'$.

Let Σ be a finite alphabet and $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$. A *finite transducer* over Σ is a 5-tuple $\tau = (Q, \Sigma_\varepsilon \times \Sigma_\varepsilon, \delta, q_0, F)$ where Q is a finite set of states, $\delta \subseteq Q \times \Sigma_\varepsilon \times \Sigma_\varepsilon \times Q$ a set of transitions, $q_0 \in Q$ an initial state, and $F \subseteq Q$ a set of final states. We call a finite transducer *length-preserving* if $\delta \subseteq Q \times \Sigma \times \Sigma \times Q$. The transition relation $\rightarrow \subseteq Q \times \Sigma^* \times \Sigma^* \times Q$ is defined as the smallest relation satisfying: (1) $q \xrightarrow{\varepsilon,\varepsilon} q$ for every $q \in Q$, (2) if $(q, a, b, q') \in \delta$, then $q \xrightarrow{a,b} q'$, and (3) if $q \xrightarrow{w,u} q'$ and $q' \xrightarrow{a,b} q''$, then $q \xrightarrow{wa,ub} q''$. Then, by abuse of notation, we identify a transducer τ with the relation $\{(w,u) \mid \exists q' \in F : q_0 \xrightarrow{w,u} q'\}$. We call a relation *regular* if it can be identified with some transducer. For a set $L \subseteq \Sigma^*$ and a relation $R \subseteq \Sigma^* \times \Sigma^*$, we denote $R(L)$ the set $\{w \in \Sigma^* \mid \exists w' \in L : (w',w) \in R\}$. Let $id \subseteq \Sigma^* \times \Sigma^*$ be the identity relation and $\circ$ the composition of relations. We define recursively the relations $\tau^0 = id$, $\tau^{i+1} = \tau \circ \tau^i$, and $\tau^* = \cup_{i=0}^{\infty} \tau^i$. Below, we suppose $id \subseteq \tau$. This implies $\tau^i \subseteq \tau^{i+1}$ for all $i \geq 0$.

Regular model checking may be used for dealing with complex safety as well as liveness properties. In this paper, however, we concentrate on verifying *reachability properties*. After all, more complex verification tasks are often reduced to reachability verification (as we later illustrate on some of our experiments too).

To verify reachability properties within the regular model checking framework, there are two basic approaches. Firstly, given a system with a transition relation modelled as a transducer τ, a regular set of initial configurations *Init* described by a finite automaton, and a set of "bad" configurations *Bad* given by another finite automaton, we may try to compute the set of reachable configurations $\tau^*(Init)$ (or an overapproximation of it) and then check whether $\tau^*(Init) \cap Bad = \emptyset$.

Problem 2.1 *Given regular sets Init and Bad and a finite transducer τ, does $\tau^*(Init) \cap Bad = \emptyset$ hold ?*

Secondly, we can try to compute the reflexive and transitive closure τ^* of the transition relation τ. Then, we may use τ^* to solve Problem 2.1. Alternatively, we may also describe the bad property using a transducer τ_{Bad}—expressing the fact that it is undesirable to be able to get from a certain configuration to another—and check whether $\tau^* \cap \tau_{Bad} = \emptyset$ (provided transducers are length-preserving).

Problem 2.2 *Given length-preserving finite transducers τ and τ_{Bad}, does $\tau^* \cap \tau_{Bad} = \emptyset$ hold ?*

Both of the above approaches have their advantages and disadvantages. Computing τ^* is often more difficult than computing just $\tau^*(Init)$. Indeed, there are cases where $\tau^*(Init)$ is regular and τ^* not. On the other hand, using τ^* may allow one to check properties that would otherwise be more difficult or impossible to check (as, e.g., checking input-output correspondence between particular elements of various unbounded structures like linked lists, queues, etc.).

To illustrate the described notions and later our method, we give in Fig. 1 the basic automata and transducers that appear in regular model checking of a simple token passing protocol. The protocol is supposed to pass a single token from the left to the right along a sequence of a parametric number of nodes—the transition relation is shown in Fig. 1(c) including the identity relation. Initially, the token is in the left-most node: Fig. 1(a). The *Bad* automaton in Fig. 1(b) says that there should never be less than or more than one token. The transducer τ_{Bad} in Fig. 1(d) states that the token should never be passed to the left. The reachability relation and the reachability set are then shown in Fig. 1(e) and 1(f).

Before proposing our new regular model checking method based on inference of regular languages that is primarily targeted at dealing with *length-preserving*

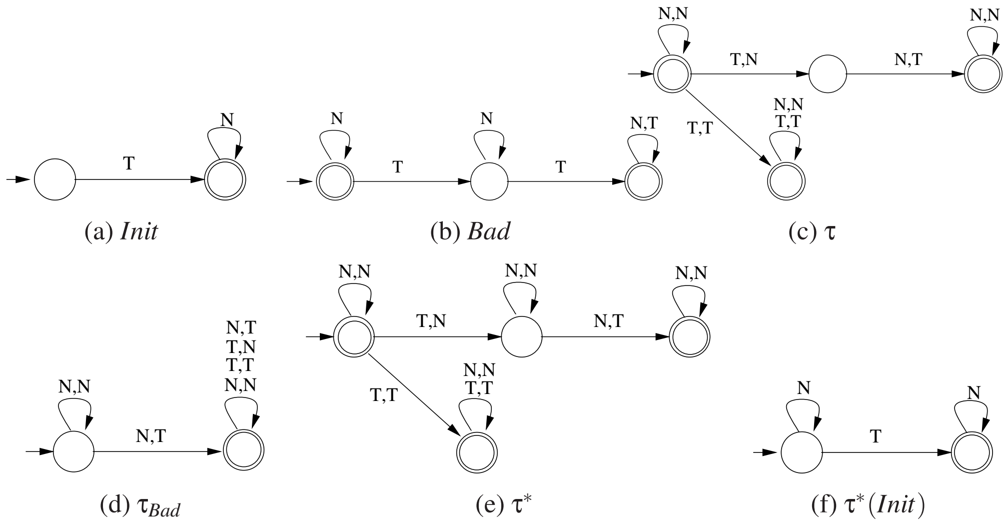

Fig. 1. Automata and transducers for regular model checking of a simple token passing

transducers, we remark that dealing with such transducers is not restrictive for verification of safety of *non-length-preserving systems* (see also [12]). This is because for finite runs, we can always replace adding and removing of symbols by rewriting special blank symbols $\perp$ added in advance into the initial configurations. More precisely, we can add self-loops labelled with $\perp,\perp$ to every state of τ (and τ_{Bad} if used), replace every ε, a by a $\perp, a$ transition, every a, ε transition by an $a, \perp$ one, and add $\perp$-labelled self-loops at every state of the automata of *Init* and *Bad*.

3 Inference of Regular Languages from Complete Training Sets

Inference of regular languages is a very active research area (see, e.g., [19,16,14,9]). Basically, the problem is to infer a language from some of its words (or words known not to belong to the language). An important notion used in the proposed algorithms is that of a training set (or sample). A *training set* $T = (T^+, T^-)$ is a pair of two disjoint sets $T^+, T^- \subseteq \Sigma^*$, where T^+ contains positive examples (words in the language to be inferred) and T^- contains negative ones. A training set $T = (T^+, T^-)$ is called *n-complete* if $T^+ \cup T^- = \Sigma^{\leq n}$. For the inference of regular languages from training sets, several different algorithms have been studied. Out of them, in this work, we propose using the so-called *Trakhtenbrot-Barzdin algorithm* (TB algorithm for short) [19] that works on n-complete training sets, which—as we show below—may be obtained in the case of regular model checking.

For length-preserving transducers, the model checking Problem 2.1 can be seen as a language inference problem in the following way: We want to compute (or at least approximate) the set $\tau^*(Init)$ for some given length-preserving transducer τ and a regular set *Init*. Since τ is length-preserving, the set $\tau^*(Init^{\leq n})$ is finite for each n and can be calculated by finitely iterating τ (recall that $id \subseteq \tau$). Furthermore, each word of length smaller or equal to n which is not in $\tau^*(Init^{\leq n})$ cannot be in $\tau^*(Init)$ either. Therefore, the sets $\tau^*(Init^{\leq n})$ and $\Sigma^{\leq n} \setminus \tau^*(Init^{\leq n})$ can be seen as sets of positive and negative examples of the language $\tau^*(Init)$ that we want to infer. precise, words of the given language up to some length and thus form an n-complete training set. For increasing n, we get more and more positive and negative examples of the language $\tau^*(Init)$—the training set is growing. Therefore, if $\tau^*(Init)$ is regular, we will eventually (we do not know when of course) obtain a training set T big enough in terms of [19], and the TB algorithm will allow us to infer $\tau^*(Init)$ from T. If $\tau^*(Init)$ is not regular, it is still possible to perhaps get an overapproximation sufficient to prove the property of interest. The same approach can also be used to try to infer the relation τ^* for length-preserving transducers by considering as the alphabet pairs of letters and calculating τ^* restricted to words of length smaller or equal to n.

3.1 The Trakhtenbrot-Barzdin Algorithm

To describe the TB algorithm and its use in regular model checking in detail, we need some more definitions. A DFA A is called *consistent* with a training set $T = (T^+, T^-)$ if $T^+ \subseteq L(A)$ and $L(A) \cap T^- = \emptyset$. A training set $T = (T^+, T^-)$ is *n-complete wrt. a DFA A* if $T^+ = L^{\leq n}(A)$. Given an n-complete training set $T = (T^+, T^-)$, we call a deterministic finite automaton $A_T = (Q, \Sigma, \delta, q_0, F)$ the *prefix-tree automaton* of T if $L(A) = T^+$, A_T has the form of a tree and does not contain any nodes with the empty language (provided $T^+ \neq \emptyset$).

We now describe a slightly modified version of the Trakhtenbrot-Barzdin algorithm which computes an inferred DFA (also called *target automaton*) with the minimal number of states consistent with a given n-complete training set. Let $T = (T^+, T^-)$ be an n-complete training set and A_T the deterministic prefix-tree automaton of T. Obviously, states of A_T must correspond to states of the target automaton $\bar{A}_T$ since it must accept all words accepted by A_T. Several different states of A_T can, however, correspond to the same state of $\bar{A}_T$. Hence, the basic idea of the algorithm is to collapse two states of A_T if this does not lead to a word of length shorter or equal to n being accepted though it is not accepted by A_T. Two states q and q' in A_T can be safely collapsed if they are *compatible*, i.e. if they are k-indistinguishable ($q \equiv_k q'$) where k is the minimum of the lengths of the subtrees starting at q and q'. Let *succ* be the function which associates to each state in A_T the successor in a breadth-first ordering. The algorithm modifies A_T by identifying compatible states.

Algorithm 1

`input:` *A_T with initial state q_0*
$q1 := q_0$;
`while` *there is a successor of q_1 in A_T* `do`
  $q_1 := succ(q_1)$; $q_2 := q_0$;
  `while` $q_1 \neq q_2$ `and not` $compatible(q_1, q_2)$ `do` $q_2 := succ(q_2)$; `od`
  `if` $q_1 \neq q_2$ `and` $compatible(q_1, q_2)$ `then` *let the transition from $father(q_1)$ to q_1 point to q_2 and erase q_1 and all its children from A_T;*
`od`
`output:` *the modified automaton A_T*

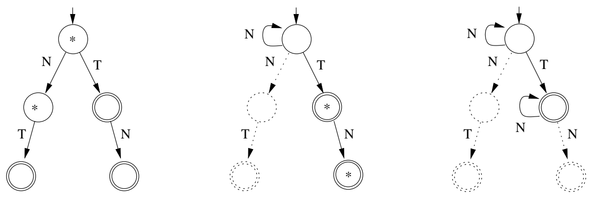

Fig. 2. A 2-complete training set and the different stages of the TB algorithm

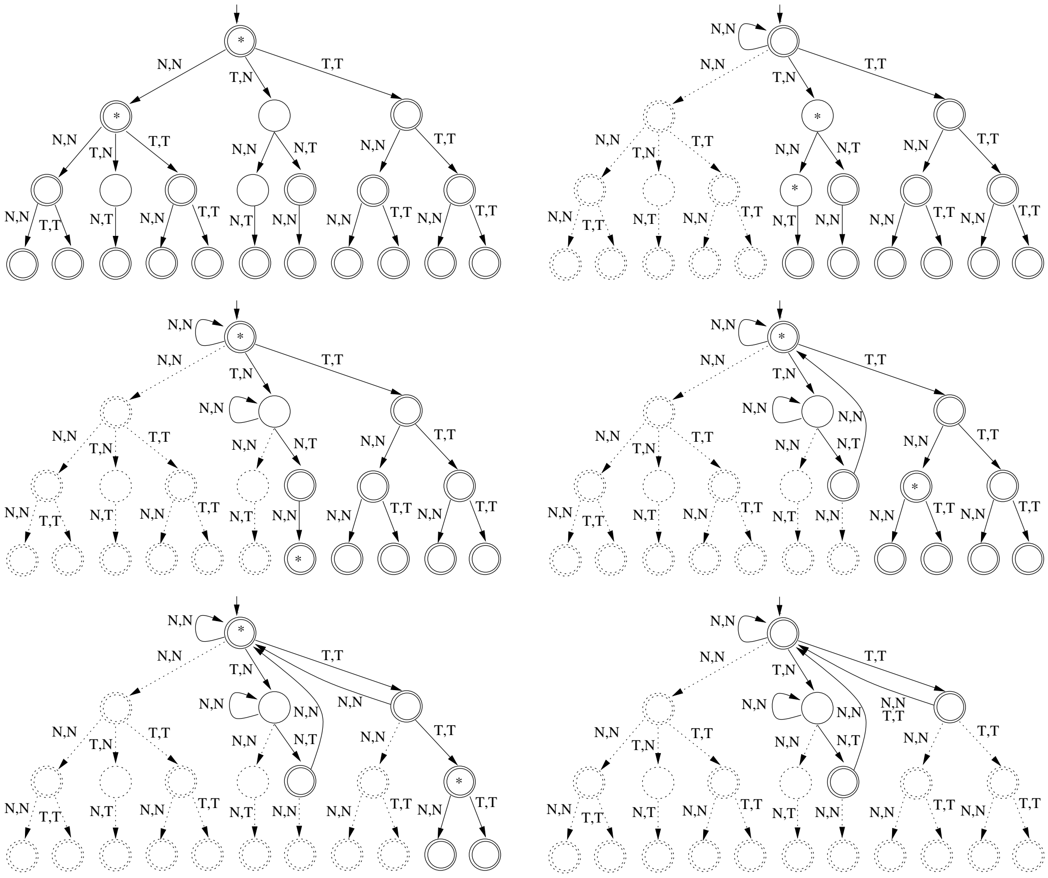

Fig. 3. A 3-complete training set and the different stages of the TB algorithm

Notice that the original algorithm [19] uses as input trees which are complete (each internal node has a son for each letter). In our setting, this is not necessary since we only consider languages and not output behaviour of automata. The algorithm has a complexity of $O(mn^2)$, where m is the size of A_T and n the size of the target automaton. In Figures 2 and 3, we give the different stages of the TB algorithm run on a 2-complete training set for $\tau^*(Init)$ and a 3-complete set for τ^* of our running example. Two collapsed compatible states at each stage are marked with $*$. Notice that in the first case, $\tau^*(Init)$ is obtained exactly as the result of the algorithm, whereas in the other case, the result is an overapproximation of τ^*.

We have the following theorem [19].

Theorem 3.1 *Let T be an n-complete training set. Algorithm 1 computes a DFA $\bar{A}_T$ with a minimal number of states consistent with T.*

Notice that there could be several different DFAs with a minimal number of states consistent with T—the output of the TB algorithm is just one of them. If all the words of the training set come from some minimal deterministic automaton A, then the TB algorithm is guaranteed to infer it from an n-complete training set if n is sufficiently big with respect to the structure of the automaton. The *degree of reconstructibility* r of an automaton A is defined as $r = d + \rho + 1$ where d is the depth and ρ the degree of distinguishability. Then we have the following theorem [19].

Theorem 3.2 *Given a minimal DFA A with degree of reconstructibility r and a training set T r-complete wrt. A, Algorithm 1 computes A (up to isomorphism).*

If A has n states, then in the worst case, $d = \rho = n-1$ and $r = 2n-1$. Therefore, the complete training set must contain exponentially (in n) many words. Fortunately, on average [19,14], r is much smaller because it can be shown that the average value of d is $Clog_{|\Sigma|}(n)$ where C is a constant depending on Σ and $\rho = log_{|\Sigma|}log_2(n)$. This means that on average, the degree of reconstructibility r is small compared to the size of the automaton and only small complete training sets (polynomial in n) are needed to reconstruct it.

4 The Model Checking Algorithm

In this section, we describe our model checking algorithm based on inference of regular languages. The idea of our algorithm is to compute bigger and bigger complete training sets coming from the language $\tau^*(Init)$, infer an automaton from them, and test whether this inferred automaton is an invariant sufficient to prove the property. As the inference algorithm, one can—in principle—use any inference algorithm based on positive and negative examples proposed in the literature (as, e.g., RPNI [16,14]). In this paper, we use the Trakhtenbrot-Barzdin algorithm discussed above because it works with a complete training set and is guaranteed to output the original automaton if given sufficiently big training sets. Below, we, however, describe our model checking algorithm in a general way.

Algorithm 2

`input`: *a length-preserving transducer* τ, *a regular set of initial configurations* $Init$ *and a regular set of bad configurations* Bad
$i := 1$; `/*` *i can be initialized differently too.* `*/`
`repeat`
  $C := \tau^*(Init^{\leq i})$; $\overline{C} := \Sigma^{\leq i} \setminus C$;
  `if` $Bad \cap C \neq \emptyset$ `then output:` *property violated;*
  $A := inference(C, \overline{C})$; $i := i+1$;
`until` $\tau(L(A)) \subseteq L(A)$ `and` $Init \subseteq L(A)$ `and` $L(A) \cap Bad = \emptyset$;
`output:` *property satisfied*

When we use the version of the TB algorithm described as Algorithm 1 for inference in Algorithm 2, the call of $inference(C, \overline{C})$ invokes Algorithm 1 with the prefix-tree automaton of C as input. Then, the computation of $\overline{C}$ is not necessary.

In our running example, to verify the property $\tau^*(Init) \cap Bad = \emptyset$, the algorithm stops for $i = 2$ (the inferred invariant is exactly $\tau^*(Init)$), and for the property $\tau^* \cap \tau_{bad} = \emptyset$, the algorithm stops for $i = 3$ with an overapproximation of τ^* (see Fig. 3).

Notice that to calculate $\tau^*(Init^{\leq i})$, one can reuse $\tau^*(Init^{\leq i-1})$ and only calculate the reachable configurations of size i. The algorithm tries bigger and bigger training sets until it terminates because it either finds a counterexample to the property of interest, or an invariant including the initial states and not intersecting the *Bad* set. The test $Init \subseteq L(A)$ is necessary because for small i, the examples generated may not suffice to reconstruct *Init*. An alternative way would be to set the initial value of i wrt. *Init*, but according to our experience, this is not always the best choice.

If $\tau^*(Init)$ is regular, the algorithm always terminates.

Theorem 4.1 *Let τ be a length-preserving transducer and Init and Bad two regular sets. If $\tau^*(Init)$ is regular, then Algorithm 2 with Algorithm 1 used as the inference algorithm always terminates.*

Proof. If $\tau^*(Init) \cap Bad \neq \emptyset$, then there exists a word $w \in \tau^*(Init) \cap Bad$ of some size n. Since τ is length-preserving, $w \in \tau^*(Init^{\leq n})$ and the algorithm terminates after at most n iterations. If $\tau^*(Init) \cap Bad = \emptyset$, then because of Theorem 3.2, the algorithm stops after at most r (the degree of reconstructibility of $\tau^*(Init)$) steps.$\square$

Notice that termination of the algorithm with the property verified means that an invariant precise enough to prove the property was inferred. In general, we cannot check whether we have inferred the exact reachability set $\tau^*(Init)$. This is clear, e.g., from the fact that for lossy channel systems, $\tau^*(Init)$ is known to be regular [7,2] but not computable [1,15]. From Theorem 4.1, we get easily the following.

Corollary 4.2 *The model checking Problem 2.1 is decidable if $\tau^*(Init)$ is regular.*

The above is not very surprising as we can give two semi-decision procedures for the problem: one looking for bigger and bigger counterexamples, the other one enumerating all regular languages and checking for invariants (as explained in [17] for FIFO-channel systems). Our algorithm provides a clever way to enumerate regular languages being candidates for an invariant.

Finally, without going into detail, it is clear that in a very similar way as above, we can deal with transition relations too.

Corollary 4.3 *The model checking Problem 2.2 is decidable if τ^* is regular.*

Remark: Algorithm 2 can easily be modified to handle non-length-preserving transducers τ too: When we calculate the fixpoint $\tau^*(Init)$, we always after each step intersect the reachable configurations with $\Sigma^{\leq n}$. In this way, the fixpoint computation will always terminate. However, the training set is not guaranteed to be complete anymore. Therefore, termination of the model-checking algorithm is not in general insured even for regular $\tau^*(Init)$. However, according to our practical experience, the method still behaves well for various concrete examples.

5 Experiments

We have implemented the ideas proposed in the paper in a prototype tool written in YAP Prolog using the FSA library [20]. As regular model checking is broadly applicable, we applied the tool to a variety of different verification tasks.

The experiments were similar to those presented in more detail in [5]. They included verification of *parametric systems* (in the form of a bit idealized parametric Bakery, Burns, Dijkstra, and Szymanski algorithms of mutual exclusion), a simple *push-down system* modelling a program with several mutually recursive procedures (the plotter control example from [10]), the alternating bit protocol as a representative of *systems with (lossy) queues*, a Petri net modelling the readers-writers problem with dynamically arising and disappearing processes that can be considered an example of a system with *unbounded counters* (whose values were encoded in unary), and a procedure for reversing linear lists as a representative of systems with *unbounded recursive data structures*. The simplified Bakery mutual exclusion algorithm was modelled in several ways: with a parametric number of processes and the values of tickets encoded by the positions of the appropriate processes in the word representing a configuration, and with a bounded number of processes (three to five) with the tickets modelled by explicit counters with values encoded either in binary (as in NDDs) or in unary.

We have mostly considered verification of *invariance properties* that can be directly handled using reachability verification. We have, however, tried dealing with some more complex properties too. The push-down example where we checked some constraint on the calling order of the procedures is an example of dealing with a bit *more complex safety properties*—to transform it to a reachability problem, we manually composed the appropriate safety automaton with the model of the system. We have also verified *communal liveness* in the Bakery example. In this case, we have manually composed the appropriate Büchi automaton with the system being verified. We have mostly considered *correct systems*, but we have as well run the tool over a *faulty version* of one of the considered systems—namely the Readers/Writers example where we omitted one of the Petri net arcs. We have mostly worked on the level of dealing with *reachability sets*, but in the example of reversing lists, we have also worked with a *reachability relation* represented by a transducer. (Using the reachability set computation, we checked that the procedure outputs a list, but using the reachability relation—restricted to reachability from initial states, i.e. to $id_{Init} \circ \tau^*$, we checked that the output list is a reversion of the input one.) Finally, in the experiments, we were trying both *forward* and *backward verification*—i.e. starting from the initial states or the "bad" states.

The results of the experiments are summarized in Table 1. For each experiment, we give the best result obtained. We say whether it was within forward or backward verification and what the initial length of the words in the sample was (the considered values were: 1, $|Q_{Bad}|$, $2|Q_{Bad}|$, $|Q_{Bad}|/2$, $|Q_{Init}|$, $2|Q_{Init}|$, and $|Q_{Init}|/2$). When we compare these results with those of [5] (which belong among the best in the field), we see that they are usually a bit slower but comparable. In one case

Table 1
Some results of experimenting with verification based on inference of regular languages

Experiment	Best setting	T [sec]	r_g [%]	Experiment	Best setting	T [sec]	r_g [%]
Bakery	Bw, $\|Q_{Bad}\|$	0.03	50	Dijkstra	Fw, $\|Q_{Bad}\|/2$	1.16	92
Bakery comm. liv.	Fw, $2\|Q_{Init}\|$	0.36	90	PDS	Bw, $\|Q_{Bad}\|$	0.04	63
Bakery counters 3P	Bw, $2\|Q_{Bad}\|$	8.69	70	Petri net/Read. Wr.	Fw, $\|Q_{Bad}\|/2$	323	90
Bakery counters 4P	Fw, $\|Q_{Bad}\|$	143	92	Faulty PN/Rd. Wr.	Fw, $2\|Q_{Init}\|$	1.48	54
Bakery 5P unary	Fw, $2\|Q_{Init}\|$	229	45	Szymanski	Fw, $\|Q_{Bad}\|$	0.76	94
ABP	Bw, $2\|Q_{Bad}\|$	0.03	50	Rev. Lists	Fw, $\|Q_{Init}\|$	1.64	90
Burns	Fw, $2\|Q_{Init}\|$	0.77	98	Rev. Lists/Transd.	Fw, $\|Q_{Init}\|/2$	40.5	69

(the ABP example), the inference method was even faster than the one of [5]. Such results are very positive taking into account that no guarantees of termination are known for the methods of [5].

Finally, in the r_g columns of Table 1, we give the percentage of time spent in generating the finite sample, which indicates that the treatment of this part of our method deserves a special attention in the future optimisations.

6 Conclusion

We introduce in this paper a new method for regular model checking based on inference of regular languages. The method iteratively computes more and more precise approximations of $\tau^*(Init)$ or τ^* that are inferred from larger and larger samples representing all reachable configurations (transductions of configurations) with length up to a certain bound that is being incrementally increased. The computation stops when the bad configurations are reached or when a safe overapproximation of $\tau^*(Init)$ or τ^* is inferred. For length-preserving transducers, the method is complete for all systems with regular $\tau^*(Init)$ or τ^*. This is a major advantage compared to other methods. At the same time, experimental results show that the method is efficient too. Our model-checking algorithm is general as every inference algorithm working with positive and negative examples can be used. In the future, we plan to try other inference algorithms [16,14] and compare their performances in our framework. An investigation of incremental inference algorithms like RPNI2 [9] could especially be interesting. They are based on refining an inference hypothesis when new positive and negative examples are provided. They could be easily used in our framework since we compute longer and longer configurations. A further optimisation could be a use of dedicated finite-state model checkers to compute the set of reachable configurations of bounded length efficiently.

References

[1] Abdulla, P., L. Boasson and A. Bouajjani, *Effective Lossy Queue Languages*, in: *Proc. of ICALP'01*, LNCS **2076** (2001).

[2] Abdulla, P., A. Bouajjani and B. Jonsson, *On-the-fly Analysis of Systems with Unbounded, Lossy Fifo Channels*, in: *Proc. of CAV'98*, LNCS **1427** (1998).

[3] Abdulla, P., J. d'Orso, B. Jonsson and M. Nilsson, *Algorithmic Improvements in Regular Model Checking*, in: *Proc. of CAV'03*, LNCS **2725** (2003).

[4] Boigelot, B., A. Legay and P. Wolper, *Iterating transducers in the Large*, in: *Proc. CAV'03*, LNCS **2725** (2003).

[5] Bouajjani, A., P. Habermehl and T. Vojnar, *Abstract Regular Model Checking*, in: *Proc. of CAV'04*, LNCS (2004).

[6] Bouajjani, A., B. Jonsson, M. Nilsson and T. Touili, *Regular Model Checking*, in: *Proc. of CAV'00*, LNCS **1855** (2000).

[7] Cécé, G., A. Finkel and S. P. Iyer, *Unreliable Channels are Easier to Verify than Perfect Channels*, Information and Computation **124** (1996), pp. 20–31.

[8] Dams, D., Y. Lakhnech and M. Steffen, *Iterating transducers*, in: *Proc. CAV'01*, LNCS **2102** (2001).

[9] Dupont, P., *Incremental Regular Inference*, in: *Grammatical Inference: Learning Syntax from Sentences*, LNAI **1147**, 1996, pp. 222–237.

[10] Esparza, J., D. Hansel, P. Rossmanith and S. Schwoon, *Efficient algorithms for model checking pushdown systems*, in: *Proc. of CAV'00*, LNCS **1855** (2000).

[11] Fribourg, L. and H. Olsen, *Reachability Sets of Parametrized Rings as Regular Languages*, in: *INFINITY workshop*, Electronic Notes in Theoretical Computer Science **9** (1997).

[12] Jonsson, B. and M. Nilsson, *Transitive Closures of Regular Relations for Verifying Infinite-State Systems*, in: *Proc. of TACAS'00*, LNCS **1785** (2000).

[13] Kesten, Y., O. Maler, M. Marcus, A. Pnueli and E. Shahar, *Symbolic Model Checking with Rich Assertional Languages*, Theoretical Computer Science **256** (2001).

[14] Lang, K. J., *Random DFA's can be Approximately Learned from Sparse Uniform Examples*, in: *Proc. of the 5th ACM Workshop on Computational Learning Theory* (1992), pp. 45–52.

[15] Mayr, R., *Undecidable Problems in Unreliable Computations*, in: *Latin American Theoretical Informatics*, 2000, pp. 377–386.

[16] Oncina, J. and P. Garcia, *Inferring Regular Languages in Polynomial Update Time*, in: *Pattern Recognition and Image Analysis*, 1992, pp. 49–61.

[17] Pachl, J., *Protocol Description and Analysis based on a State Transition Model with Channel Expressions*, in: *Protocol Verification, Specification, and Testing VII*, 1987.

[18] Touili, T., *Widening Techniques for Regular Model Checking*, ENTCS **50** (2001).

[19] Trakhtenbrot, B. A. and Y. A. Barzdin, "Finite Automata: Behavior and Synthesis," North-Holland, 1973.

[20] van Noord, G., *FSA6.2* (2004), URL: http://odur.let.rug.nl/~vannoord/Fsa/.

[21] Vardhan, A., K. Sen, M. Viswanathan and G. Agha, *Learning to Verify Safety Properties*, in: *Proc. of ICFEM'04*, 2004, accepted for publication.