A Survey of Regular Model Checking

Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena

Dept. of Information Technology, P.O. Box 337, S-751 05 Uppsala, Sweden
{parosh,bengt,marcusn,mayanks}@it.uu.se

Abstract. Regular model checking is being developed for algorithmic verification of several classes of infinite-state systems whose configurations can be modeled as words over a finite alphabet. Examples include parameterized systems consisting of an arbitrary number of homogeneous finite-state processes connected in a linear or ring-formed topology, and systems that operate on queues, stacks, integers, and other linear data structures. The main idea is to use regular languages as the representation of sets of configurations, and finite-state transducers to describe transition relations. In general, the verification problems considered are all undecidable, so the work has consisted in developing semi-algorithms, and decidability results for restricted cases. This paper provides a survey of the work that has been performed so far, and some of its applications.

1 Introduction

A significant research effort is currently being devoted to extending the applicability of algorithmic verification to parameterized and infinite-state systems, using approaches based on abstraction, deductive techniques, decision procedures, etc. One major approach is to extend the paradigm of symbolic model checking [BCMD92] to new classes of models by an appropriate symbolic representation; examples include timed automata, systems with unbounded communication channels, Petri nets, and systems that operate on integers and reals.

Regular model checking is such an extension, in which sets of states and transition relations are represented by regular sets, typically over finite or infinite words or tree structures. Most work has considered models whose configurations can be represented as finite words of arbitrary length over a finite alphabet. This includes parameterized systems consisting of an arbitrary number of homogeneous finite-state processes connected in a linear or ring-formed topology, and systems that operate on queues, stacks, integers, and other linear data structures. Regular model checking was advocated by Kesten et al. [KMM+01] and by Boigelot and Wolper [WB98], as a uniform framework for analyzing several classes of parameterized and infinite-state systems. The idea is that regular sets will provide an efficient representation of infinite state spaces, and play a role similar to that played by Binary Decision Diagrams (BDDs) for symbolic model checking of finite-state systems. One can also exploit automata-theoretic algorithms for manipulating regular sets. Such algorithms have been successfully implemented, e.g., in the Mona [HJJ+96] system.

A generic task in symbolic model checking is to compute properties of the set of reachable states, in order to verify safety properties. For finite-state systems this is typically done by state-space exploration, but for infinite-state systems this procedure terminates only if there is a bound on the distance (in number of transitions) from the initial configurations to any reachable configuration. An analogous observation holds if we perform a reachability analysis backwards, by iteration-based methods [CES86,QS82] from a set of "unsafe" configurations. A parameterized or infinite-state system does not have such a bound, and any non-trivial model checking problem is undecidable. In contrast to deductive application of systems like Mona [BK98], the goal in regular model checking is to verify system properties algorithmically. An important challenge is therefore to devise so-called acceleration techniques, which calculate the effect of arbitrarily long sequences of transitions. This problem has been addressed in regular model checking [JN00,BJNT00,AJNd02]. In general, the effect of acceleration is not computable. However, computability have been obtained for certain classes [JN00]. Analogous techniques for computing accelerations have successfully been developed for several classes of parameterized and infinite-state systems, e.g., systems with unbounded FIFO channels [BG96,BGWW97,BH97,ABJ98], systems with stacks [BEM97,Cau92,FWW97,ES01], and systems with counters [BW94,CJ98].

In this paper, we survey the available work on regular model-checking. The use of regular sets to model and specify systems is discussed in Section 2. Techniques for computing invariants and reachable loops are surveyed in Section 3. Finally, some extensions of the paradigm are discussed in Section 4.

2 Framework

Model checking is concerned with automated analysis of transition systems, each consisting of

- a set of *configurations* (or states), some of which are *initial*, and
- a *transition relation*, which is a binary relation on the set of configurations.

The configurations represent possible "snapshots" of the system state, and the transition relation describes how these can evolve over time. Most work on model checking assumes that the set of configurations is finite, but significant effort is underway to develop model checking techniques for transition systems with infinite sets of configurations.

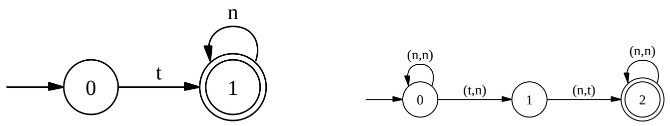
In its simplest form, the regular model checking framework represents a transition system as follows.

- A *configuration* (state) of the system is a word over a finite alphabet Σ.
- The set of *initial configurations* is a regular set over Σ.
- The *transition relation* is a regular and length-preserving relation on Σ^*. It is represented by a finite-state transducer over $(\Sigma \times \Sigma)$, which accepts all words $(a_1, a_1') \cdots (a_n, a_n')$ such that $(a_1 \cdots a_n \ , \ a_1' \cdots a_n')$ is in the transition relation. Sometimes, the transition relation is given as a union of a finite number of relations, each of which is called an *action*.

Given a transducer T, we often abuse notation and use T also to denote the relation defined by the transducer. For a set S of configurations and a binary relation R on configurations, let $S \circ R$ denote the set of configurations w such that $w' \ R \ w$ for some $w' \in S$, let R^+ denote the transitive closure of R and R^* denote the reflexive transitive closure of R. Let S^2 denote the set of pairs of elements in S.

In the regular model checking framework it is possible to model parameterized systems with linear or ring-shaped topologies, e.g., by letting each position in the word model the state of a system component. It is also possible to model programs that operate on linear unbounded data structures such as queues, stacks, integers, etc. For instance, a stack can be modeled by letting each position in the word represent a position in the stack. The restriction to length-preserving transducers implies that we cannot dynamically "create" new stack positions. Therefore the stack should initially contain an arbitrary but bounded number of empty stack positions, which are "statically allocated". We can then faithfully model all finite computations of the system, by initially allocating sufficiently many empty stack positions. However, it may not be possible to model faithfully all infinite computations of the system. Thus, the restriction to length-preserving transducers introduces no limitations for analyzing safety properties, but may incur restrictions on the ability to specify and verify liveness properties of systems with dynamically allocated data structures.

2.1 Examples



(a) Automaton for I (b) Transducer T

Fig. 1. Initial set of states and transition relation

In Figure 1 we consider a *token passing protocol*: a simple parameterized system consisting of an arbitrary (but finite) number of processes organized in a linear fashion. Initially, the left-most process has the token. In each step, the process currently having the token passes it to the right. A configuration of the system is a word over the alphabet $\{t, n\}$, where t represents that the process has the token, and n represents not having it. For instance, the word $nntnn$ represents a configuration of a system with five processes where the third process has the token. The set of initial states is given by the regular expression tn^* (Figure 1(a)). The transition relation is represented by the transducer in Figure 1(b).

For instance, the transducer accepts the word $(n,n)(n,n)(t,n)(n,t)(n,n)$, representing the pair ($nntnn$, $nnntn$) of configurations where the token is passed from the third to the fourth process.

As a second example, we consider a system consisting of a finite-state process operating on one unbounded FIFO channel. Let Q be the set of control states of the process, and let M be the (finite) set of messages which can reside inside the channel. A configuration of the system is a word over the alphabet $Q \cup M \cup \{e\}$, where the *padding symbol* e represents an empty position in the channel. For instance the word $q_1 e m_3 m_1 e e$ corresponds to a configuration where the process is in state q_1 and the channel (of length four) contains the messages m_3 and m_1 in this order. The set of configurations of the system can thus be described by the regular expression $Qe^*M^*e^*$.

By allowing arbitrarily many padding symbols e, one can model channels of arbitrary but bounded length. As an example, the action where the process sends the message m to the channel and changes state from q_1 to q_2 is modeled by the transducer in Figure 2. In the figure, "M" is used to denote any message in M.

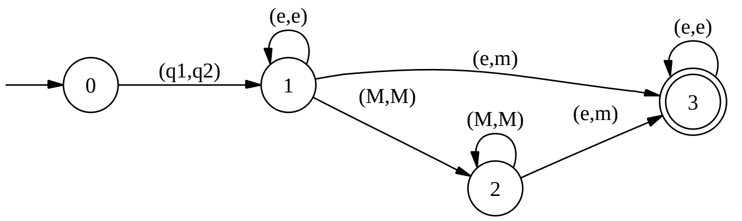


Fig. 2. Transducer for sending a message m to the channel

2.2 Verification Problems

We will consider two types of verification problems in this paper.

The first problem is verification of *safety properties*. A safety property is of form "bad things do not happen during system execution". A safety property can be verified by solving a *reachability* problem. Formulated in the regular model checking framework, the corresponding problem is the following: given a set of initial configurations I, a regular set of *bad configurations* B and a transition relation specified by a transducer T, does there exist a path from I to B through the transition relation T? This amounts to checking whether $(I \circ T^*) \cap B = \emptyset$. The problem can be solved by computing the set $Inv = I \circ T^*$ and checking whether it intersects B.

The second problem is verification of *liveness properties*. A liveness property is of form "a good thing happens during system execution". Often, liveness properties are verified using fairness requirements on the model, which can state that certain actions must infinitely often be either disabled or executed. Since, by the restriction to length-preserving transducers, any infinite system execution can only visit a finite set of configurations, the verification of a liveness property can be reduced to a *repeated reachability* problem. The repeated reachability problem asks, given a set of initial configurations I, a set of *accepting configurations* F and a transition relation T, whether there exist an infinite computation from I through T that visits F infinitely often? By letting F be the configurations where the fairness requirement is satisfied, and by excluding states where the "good thing" happens from T, the liveness property is satisfied if and only if the repeated reachability problem is answered negatively.

Since the transition relation is length-preserving, and hence each execution can visit only a finite set of configurations, the repeated reachability problem can be solved by checking whether there exists a reachable loop containing some configuration from F. This can be checked by computing $(Inv \cap F)^2 \cap Id$ and checking whether this relation intersects T^+. Here Id is the identity relation on the set of configurations, and $Inv = I \circ T^*$ as before.

Sets like $I \circ T^*$ and relations like T^+ are in general not regular or even computable (note that T could model the computation steps of a Turing machine). Even if they are regular, they are sometimes not effectively computable. In these cases, the above verification problems cannot be solved by the proposed techniques. Therefore, a main challenge in regular model checking is to design semi-algorithms which successfully compute such sets and relations for as many examples as possible. In Section 3, we briefly survey some techniques that have been developed for this purpose.

2.3 A Specification Logic

The translation from a problem of verifying liveness under fairness requirements to a repeated reachability problem can be rather tricky. One way to make the task easier is to provide an intuitive syntax for modeling and specification, which can be automatically translated to repeated reachability problems, in analogy with the way that linear-time temporal logic formulas are translated to Büchi automata [VW86].

A logic LTL(MSO) was proposed for regular model checking in [AJN+04]. It uses a MSO (monadic second-order logic) over finite words to specify regular sets, and LTL to specify temporal properties. The problem of model checking a formula in LTL(MSO) can be automatically translated into a repeated reachability problem [AJN+04].

The logic LTL(MSO) combines (under certain restrictions) temporal operators of LTL [KPR98], including $\Box$ (*always*) and $\Diamond$ (*eventually*), and MSO quantification over *positions* (first-order) and *sets of positions* (second-order). Models of LTL(MSO) formulas are sequences of configurations (i.e., words), where the first-order position variables denote positions in configurations, and the second-order variables denote sets of positions. For instance, if $\varphi(i)$ is a formula which specifies a temporal property at position i in the word, then the formula $\forall i \Diamond \varphi(i)$ specifies that $\varphi(i)$ eventually holds at each position in the word.

In LTL(MSO), one can represent the configuration of a system by *configuration predicates*, which can be seen as Boolean arrays indexed by positions. For instance, in the token passing example, we can introduce a configuration predicate t, where the atomic formula $t[i]$ is interpreted as "the process at position i has the token", and $t'[i]$ as "the process at position i will have the token in the next time step".

Example Our running example, token passing, is modeled in LTL(MSO) below following the style of TLA [Lam94], where the system and the property of interest are both specified by formulas. The local states of processes are represented by a configuration predicate t – for every i, we have that $t[i]$ is true if and only if process i has the token. The set of initial states is modeled by **initial**, where only the first process has the token. The transition relation where the token is passed from position i to position $i+1$ is modeled by $\mathbf{pass}(i)$. Finally, the entire system model is specified by **system**. The system actions are "one process passes the token, or all processes idle". Models of this formula correspond to runs of the system.

$$\begin{aligned}
\mathbf{initial} &= \forall i\ (t[i] \ \leftrightarrow\ i = 0) \\
\mathbf{idle}(i) &= t[i] \ \leftrightarrow\ t'[i] \\
\mathbf{pass}(i) &= (t[i] \wedge \neg t'[i]) \ \wedge\ (\neg t[i+1] \wedge t'[i+1]) \wedge \\
&\quad\ \forall k\ ((k \neq i+1 \wedge k \neq i) \ \rightarrow\ \mathbf{idle}(k)) \\
\mathbf{system} &= \mathbf{initial} \ \wedge\ \Box(\exists i\ \mathbf{pass}(i) \vee \forall i\ \mathbf{idle}(i))
\end{aligned}$$

An example of a safety property for this system is "two different processes may not have the token at the same time":

$$\mathbf{safety} = \Box\neg\ \exists i, j\ (i \neq j \ \wedge\ t[i] \ \wedge\ t[j])$$

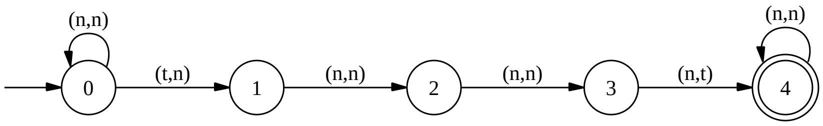
In order to specify termination ("the last process eventually gets the token") we add a fairness constraint for the token passing action. For an action α, let enabled(α) represent the set of states where the action α can be taken. enabled(α) can be expressed in the logic, using an existential quantification of the primed configuration predicates in α.

$$\begin{aligned}
\mathbf{fairness} &= \forall i\ \Box\Diamond(\mathbf{pass}(i) \vee \neg \mathrm{enabled}(\mathbf{pass}(i))) \\
\mathbf{termination} &= \Diamond\exists i\ (t[i] \ \wedge\ \forall j\ \neg(j = i+1))
\end{aligned}$$

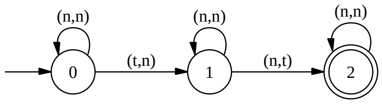
To check that the algorithm satisfies the safety property, we translate the property **system** $\wedge$ $\neg$**safety** to a reachability problem. To check that the algorithm satisfies the liveness property, we translate the property **system** $\wedge$ **fairness** $\wedge$ $\neg$**termination** to a repeated reachability problem.

3 Algorithms

In Section 2, we stated a verification problem as that of computing a representation of $I \circ T^*$ (or T^+) for some transition relation T and some set of configurations I. In some cases we also have a set of *bad* configurations B and we want to check whether $I \circ T^* \cap B \neq \emptyset$. Algorithms for regular model checking are usually based on starting from I and repeatedly applying T. As a running illustration, we will consider the problem of computing the transitive closure T^+ for the transducer in Figure 1(b). A first attempt is to compute T^n, the composition of T with itself n times for $n = 1, 2, 3, \cdots$. For example, T^3 is the transition relation where the token gets passed three positions to the right. Its transducer is given below.



A transducer for T^+ is one where the token gets passed an arbitrary number of times, given below.

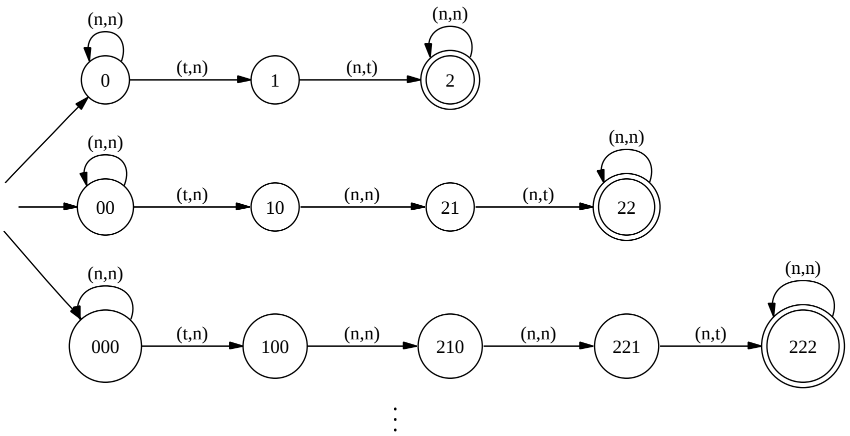


The challenge is to derive the above transducer algorithmically. Obviously, it cannot be done naively by simply computing the approximations T^n for $n = 1, 2, 3, \cdots$, since this will not converge. Some acceleration or widening techniques must be developed that compute a representation of T^+ by other means. In this section, we present some techniques developed in the literature for that purpose.

3.1 Quotienting

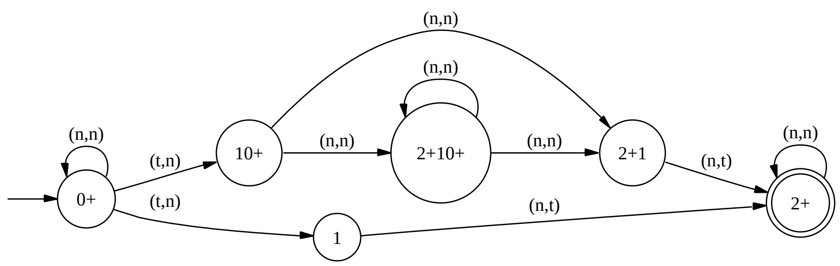
Several techniques in the literature are based on suitable *quotienting* of transducers that represent approximations of T^n for some value(s) of n. This involves finding an *equivalence relation* $\simeq$ on the states of approximations, and to merge equivalent states, obtaining a quotient transducer. For instance, in the transducer that represents T^3 above, we can define the states 1, 2, and 3 to be equivalent. By merging them, we obtain the transducer $T^3/\simeq$ which in this example happens to be equivalent to T^+.

One problem is that quotienting in general increases the language accepted by a transducer: $\mathcal{L}(T^n) \subseteq \mathcal{L}(T^n/\simeq)$, usually with strict inclusion. This problem was resolved in [AJNd02,BJNT00,DLS01,AJMd02] by characterizing equivalence relations $\simeq$ such that T^+ is equivalent to $(T/\simeq)^+$ for any transducer T, i.e., the quotienting does not increase the transitive closure of the transducer. To explain the idea, let us first build explicitly a transducer for T^+ as the union of transducers T^n for $n = 1, 2, 3, \cdots$. Each state of T^n is labeled with a sequence of states from T, resulting from the product construction using n copies of T. The result is called the *history transducer*. The history transducer corresponding to Figure 1(b) is shown below.



Recall minimization algorithms for automata. They are based on building a *forward* bisimulation $\simeq_F$ on the states, and then carry out minimization by quotienting. For instance, in the above history transducer, all states with names of form 2^i1 for any $i \geq 0$ are forward bisimilar. Analogously, we can find a backward bisimulation $\simeq_B$. For instance, all states with names of form 10^i, $i \geq 0$, are backward bisimilar. Dams et al. [DLS01] showed how to combine a forward $\simeq_F$ and a backward bisimulation $\simeq_B$ into an equivalence relation $\simeq$ which preserves the transitive closure of the transducer. In [AJNd03], this result was generalized to consider *simulations* instead of bisimulations. The simulations can be obtained by computing properties of the original automaton T (as in [AJNd02,AJNd03]), or on successive approximations of T^n (as in [DLS01]).

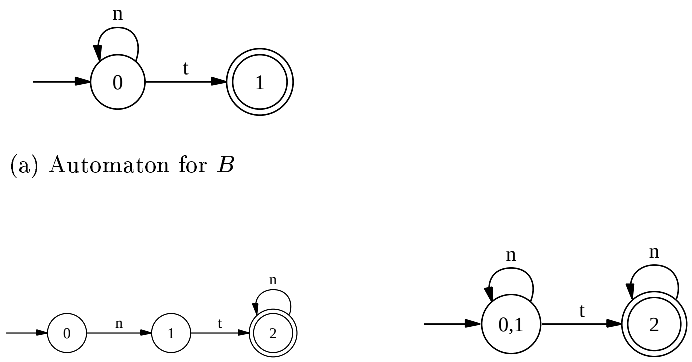
From the results in [AJNd03] it follows for the above history transducer that the states with names in 2^i1 can be merged for $i \geq 1$, and the same holds for 10^i. The equivalence classes for that transducer would be 2^+, 0^+, 10^+, 2^+1 and 2^+10^+. Hence, it can be quotiented to the following transducer, which can be minimized to the three-state representation shown earlier.



3.2 Abstraction

In recent work, Bouajjani et al. [BHV04] apply *abstraction* techniques to automata that arise in the iterative computation of $I \circ T^*$. When computing the sequence $I, I \circ T, I \circ T^2, I \circ T^3, \cdots$ the automata that arise in the computation may all be different or may be very large and contain information that is not relevant for checking whether $I \circ T^*$ has a nonempty intersection with the set of bad configurations B. Therefore, each iterate $I \circ T^n$ is abstracted by quotienting under some equivalence relation $\simeq$. In contrast to the techniques of [AJNd02,BJNT00,DLS01,AJMd02], the abstraction does not need to preserve the language accepted, i.e., $(I \circ T^n)/\simeq$ can be any over-approximation of $I \circ T^n$ or even of $I \circ T^*$. The procedure calculates the sequence of approximations of form $(((I \circ T)/\simeq) \circ T)/\simeq \cdots$. Convergence to a limit T^{lim} can be ensured by choosing $\simeq$ to have finite index.

If now $T^{lim} \cap B = \emptyset$, we can conclude (by $\mathcal{L}((I \circ T^*)) \subseteq \mathcal{L}(T^{lim})$) that $I \circ T^*$ has an empty intersection with B. Otherwise, we try to trace back the computation from B to I. If this succeeds, a counterexample has been found, otherwise the abstraction must be refined by using a finer equivalence relation, from which a more exact approximation T^{lim} can be calculated, etc.



(a) Automaton for B

(b) An automaton A

(c) The abstract version of A

Fig. 3. Applying abstraction

The technique relies on defining suitable equivalence relations. One way is to use the automaton for B. We illustrate this on the token passing example. Suppose that B is given by the automaton in Fig 3(a), denoting that the last process has the token. Each state q in an automaton A has a *post language* $\mathcal{L}(A, q)$ which is the set of words accepted starting from that state. For example, in the automaton for B we have $\mathcal{L}(B, 0) = n^*t$ and $\mathcal{L}(B, 1) = \{\epsilon\}$. The post languages are used to define $\simeq$, such that $q \simeq q'$ holds if for all states r of B we have $\mathcal{L}(A, q) \cap \mathcal{L}(B, r) = \emptyset$ exactly when $\mathcal{L}(A, q') \cap \mathcal{L}(B, r) = \emptyset$. Each equivalence class of $\simeq$ can be represented by a Boolean vector indexed by states of B, which is true on position s exactly when the equivalence class members have nonempty intersection with $\mathcal{L}(B, s)$. This is one way to get a finite index equivalence relation.

We show an example of an automaton A in Fig 3(b) with its corresponding abstract version in Fig 3(c). Considering the states of A, we observe that the post languages of states 0 and 1 both have a nonempty intersection with the post language n^*t and an empty intersection with the post language containing the empty string. The post language of state 2 have an empty intersection with n^*t and an nonempty intersection with the post language containing the empty string.

If a *spurious* counterexample is found, i.e. a counterexample occurring when quotienting with an equivalence $\simeq$, but not in the original system, we need to refine the equivalence and start again. Automata representing parts of the counterexample can be used, in the same way as the automaton B above, to define an equivalence. In [BHV04], the equivalence is refined by using *both B and* automata representing parts of the counterexample. This prevents the same counterexample from occurring twice. Using abstraction can potentially greatly reduce the execution time, since we only need to verify that we cannot reach B and therefore it may be that less information about the structure of $I \circ T^*$ needs to be stored.

3.3 Extrapolation

Another technique for calculating $I \circ T^*$ is to speed up the iterative computation by *extrapolation* techniques that try to guess the limit. The idea is to detect a repeating pattern – a regular growth – in the iterations, from which one guesses the effect of arbitrarily many iterations. The guess may be exactly the limit, or an approximation of it.

In [BJNT00,Tou01], the extrapolation is formulated in terms of rules for guessing $I \circ T^*$ from observed growth patterns among the approximations $I, I \circ T, I \circ T^2, \cdots$. Following Bouajjani et al. [BJNT00], if I is a regular expression ρ which is a concatenation of form $\rho = \rho_1 \cdot \rho_2$, and in the successive approximations we observe a growth of form $(\rho_1 \cdot \rho_2) \circ T = \rho_1 \cdot \Lambda \cdot \rho_2$ for some regular expression Λ, then the guess for the limit $\rho \circ T^*$ is $\rho_1 \cdot \Lambda^* \cdot \rho_2$. Touili [Tou01] extends this approach to more general situations. One of these is when ρ is a concatenation of form $\rho_1 \cdot \ldots \cdot \rho_n$ and

$$(\rho_1 \cdot \ldots \cdot \rho_n) \circ T = \bigcup_{i=1}^{n-1} \rho_1 \cdot \ldots \cdot \rho_i \cdot \Lambda_i \cdot \rho_{i+1} \cdot \ldots \cdot \rho_n$$

The guess for the limit $\rho \circ T^*$ is in this case

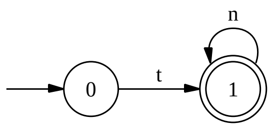
$$\rho_1 \cdot \Lambda_1^* \cdot \rho_2 \cdot \Lambda_2^* \cdot \ldots \cdot \Lambda_{n-1}^* \cdot \rho_n$$

For example, if $\rho = a^*ba^*$ and T is a relation which changes an a to a c, then $\rho \circ T$ is $a^*ca^*ba^* \cup a^*ba^*ca^*$ (i.e., each step adds either ca^* to the left of b or a^*c to the right). The above rule guesses the limit $\rho \circ T^*$ to be $a^*(ca^*)^*b(a^*c)^*a^*$. Touili also suggests other, more general, rules.

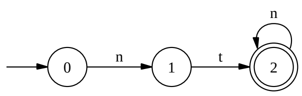
Having formed a guess ρ' for the limit, we apply a *convergence test* which checks whether $\rho' = (\rho' \circ T) \cup \rho$. If it succeeds, we can conclude that $\rho \circ T^* \subseteq \rho'$. The work in [BJNT00] and [Tou01] also provide results which state that under some additional conditions, we can in fact conclude that $\rho \circ T^* = \rho'$, i.e., that ρ' is the exact limit.

Boigelot et al. [BLW03] extend the above techniques by considering growth patterns for subsequences of $I, I \circ T, I \circ T^2, \cdots$, consisting of infinite sequences of *sample points*, noting that the union of the approximations in any such subsequence is equal to the union of the approximations in the full sequence. They apply this idea to iterate a special case of relations, *arithmetic transducers*, which operate on binary encodings of integers, and give a sufficient criterion for exact extrapolation.

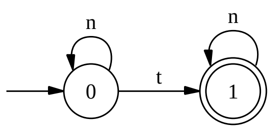
We illustrate these approaches, using our token passing example. From the initial set $\rho_I = tn^*$, we get $\rho_I \circ T = ntn^*$, $\rho_I \circ T^2 = nntn^*$, $\rho_I \circ T^3 = nnntn^*$, and so on. The methods above detect the growth $\rho_I \circ T = n \cdot \rho_I$, and guess that the limit is n^*tn^*. In this case, the completeness results of [BJNT00,Tou01] allow to conclude that the guessed limit is exact.



(a) Automaton for ρ_I



(b) Automaton for $\rho_I \circ T$



(c) Extrapolated automaton

Fig. 4. Extrapolating token passing

4 Further Directions

In previous sections, we have presented main techniques in regular model checking for the case where system configurations are modeled as finite words, and transition relations are modeled as length-preserving transducers. In this section, we briefly mention some work where these restrictions are lifted.

Non-Length-Preserving Transducers. Lifting the restriction of length-preservation from transducers allows to model more easily dynamic data structures and parameterized systems of processes with dynamic process creation. The techniques have been extended, see, e.g., [DLS01,BLW03].

Infinite Words. The natural extension to modeling systems by infinite words has been considered by Boigelot et al. [BLW04], having the application to real arithmetic in mind. Regular sets and transducers must then be represented by Büchi automata. To avoid the high complexity of some operations on Büchi automata, the approach is restricted to sets that can be defined by weak deterministic Büchi automata.

Finite Trees. Regular sets of trees can in principle be analyzed in the same way as regular sets of words, as was observed also in [KMM+01]. With some complications, similar techniques can be used for symbolic verification [AJMd02,BT02]. Some techniques have been implemented and used to verify simple token-passing algorithms [AJMd02], or to perform data-flow analysis for parallel programs with procedures [BT02].

Context Free Languages. Fisman and Pnueli [FP01] use representations of context-free languages to verify parameterized algorithms, whose symbolic verification require computation of invariants that are non-regular sets of finite words. The motivating example is the Peterson algorithm for mutual exclusion among n processes [PS81].

References

[ABJ98] Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *Proc. 10^{th} Int. Conf. on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318, 1998.

[AJMd02] Parosh Aziz Abdulla, Bengt Jonsson, Pritha Mahata, and Julien d'Orso. Regular tree model checking. In *Proc. 14^{th} Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, 2002.

[AJN+04] P.A. Abdulla, B. Jonsson, Marcus Nilsson, Julien d'Orso, and M. Saksena. Regular model checking for MSO + LTL. In *Proc. 16^{th} Int. Conf. on Computer Aided Verification*, 2004. to appear.

[AJNd02] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d'Orso. Regular model checking made simple and efficient. In *Proc. CONCUR 2002, 13^{th} Int. Conf. on Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 116–130, 2002.

[AJNd03] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d'Orso. Algorithmic improvements in regular model checking. In *Proc. 15^{th} Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 236–248, 2003.

[BCMD92] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *Proc. Intern. Conf. on Concurrency Theory (CONCUR'97)*. LNCS 1243, 1997.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In Alur and Henzinger, editors, *Proc.* 8^{th} *Int. Conf. on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12. Springer Verlag, 1996.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. of the Fourth International Static Analysis Symposium*, Lecture Notes in Computer Science. Springer Verlag, 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. In *Proc. ICALP '97,* 24^{th} *International Colloquium on Automata, Languages, and Programming*, volume 1256 of *Lecture Notes in Computer Science*, 1997.

[BHV04] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *Proc.* 16^{th} *Int. Conf. on Computer Aided Verification*, 2004. to appear.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In Emerson and Sistla, editors, *Proc.* 12^{th} *Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer Verlag, 2000.

[BK98] D.A. Basin and N. Klarlund. Automata based symbolic reasoning in hardware verification. *Formal Methods in Systems Design*, 13(3):255–288, November 1998.

[BLW03] Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large. In *Proc.* 15^{th} *Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235, 2003.

[BLW04] Bernard Boigelot, Axel Legay, and Pierre Wolper. Omega regular model checking. In K. Jensen and A. Podelski, editors, *Proc. TACAS '04,* 10^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *Lecture Notes in Computer Science*, pages 561–575. Springer Verlag, 2004.

[BT02] Ahmed Bouajjani and Tayssir Touili. Extrapolating Tree Transformations. In *Proc.* 14^{th} *Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, 2002.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc.* 6^{th} *Int. Conf. on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67. Springer Verlag, 1994.

[Cau92] Didier Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, Nov. 1992.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specification. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, April 1986.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *CAV'98*. LNCS 1427, 1998.

[DLS01] D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, 2001.

[ES01] J. Esparza and S. Schwoon. A BDD-based model checker for recursive programs. In Berry, Comon, and Finkel, editors, *Proc.* 13^{th} *Int. Conf. on*

Computer Aided Verification, volume 2102 of *Lecture Notes in Computer Science*, pages 324–336, 2001.

[FP01] Dana Fisman and Amir Pnueli. Beyond regular model checking. In *Proc. 21th Conference on the Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, December 2001.

[FWW97] A. Finkel, B. Willems, , and P. Wolper. A direct symbolic approach to model checking pushdown systems (extended abstract). In *Proc. Infinity'97, Electronic Notes in Theoretical Computer Science*, Bologna, Aug. 1997.

[HJJ+96] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Proc. TACAS '95, 1^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 of *Lecture Notes in Computer Science*, 1996.

[JN00] Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In S. Graf and M. Schwartzbach, editors, *Proc. TACAS '00, 6^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, 2000.

[KMM+01] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256:93–112, 2001.

[KPR98] Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. In *Proc. ICALP '98, 25^{th} International Colloquium on Automata, Languages, and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 1–16. Springer Verlag, 1998.

[Lam94] L. Lamport. The temporal logic of actions. *ACM Trans. on Programming Languages and Systems*, 16(3):872–923, May 1994.

[PS81] G.E. Peterson and M.E. Stickel. Myths about the mutal exclusion problem. *Information Processing Letters*, 12(3):115–116, June 1981.

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *5th International Symposium on Programming, Turin*, volume 137 of *Lecture Notes in Computer Science*, pages 337–352. Springer Verlag, 1982.

[Tou01] T. Touili. Regular Model Checking using Widening Techniques. *Electronic Notes in Theoretical Computer Science*, 50(4), 2001. Proc. Workshop on Verification of Parametrized Systems (VEPAS'01), Crete, July, 2001.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS '86, 1^{st} IEEE Int. Symp. on Logic in Computer Science*, pages 332–344, June 1986.

[WB98] Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. 10th Int. Conf. on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 88–97, Vancouver, July 1998. Springer Verlag.