FI MU

Faculty of Informatics
Masaryk University Brno

# Distributed Qualitative LTL Model Checking of Markov Decision Processes

by

Jiří Barnat
Luboš Brim
Ivana Černá
Milan Češka
Jana Tůmová

FI MU Report Series FIMU-RS-2006-04

# Distributed Qualitative LTL Model Checking of Markov Decision Processes[*]

Jiří Barnat
Faculty of Informatics, MU Brno
Botanická 68a, 602 00 Brno,
Czech Republic
barnat@fi.muni.cz

Luboš Brim
Faculty of Informatics, MU Brno
Botanická 68a, 602 00 Brno,
Czech Republic
brim@fi.muni.cz

Ivana Černá
Faculty of Informatics, MU Brno
Botanická 68a, 602 00 Brno,
Czech Republic
cerna@fi.muni.cz

Milan Češka
Faculty of Informatics, MU Brno
Botanická 68a, 602 00 Brno,
Czech Republic
xceska@fi.muni.cz

Jana Tůmová
Faculty of Informatics, MU Brno
Botanická 68a, 602 00 Brno,
Czech Republic
xtumova@fi.muni.cz

September 4, 2006

**Abstract**

Probabilistic processes are used to model concurrent programs that exhibit uncertainty. The state explosion problem for probabilistic systems is more critical than in the non-probabilistic case. In the paper we propose a cluster-based algorithm for qualitative LTL model checking of finite state Markov decision processes. We use

[*]This work has been partially supported by the Grant Agency of Czech Republic grant No. 201/06/1338 and the Academy of Sciences grant No. 1ET408050503.

the automata approach which reduces the model checking problem to the question of existence of an accepting end component. The algorithm uses repeated reachability which systematically eliminates states that cannot belong to any accepting end component. A distinguished feature of the distributed algorithm is that its complexity meets the complexity of the best known sequential algorithm.

## 1 Introduction

Probabilistic systems like Markov chains and Markov decision processes provide a reasonable semantics for systems that exhibit uncertainty. A number of qualitative and quantitative model checking algorithms for finite state probabilistic systems have been proposed [18, 29, 11, 1, 12, 20, 13]. In a qualitative setting it is checked whether a property holds with probability 0 or 1; in a quantitative setting it is verified whether the probability for a certain property meets a given lower or upper bound.

For probabilistic systems the state explosion problem is more critical than in the non-probabilistic case. Several methods that have been developed for non-probabilistic systems to avoid the state explosion were adapted to probabilistic systems. For branching time logics these are the symbolic approach [4] implemented in the model checker PRISM [22, 19] and the MDP model checker RAPTURE [8] which uses an iterative abstraction refinement. For linear time logic the most prominent partial order approach has been recently adapted as well [5, 3] and implemented in the verification tool LiQuor [9].

Over the past decade, many techniques using distributed and/or parallel processing have been proposed to combat the computational complexity of non-probabilistic verification, model checking in particular. However, not much has been done in applying these techniques to the verification and analysis of probabilistic systems. A notable exception is the work on parallelizing the symbolic model checker PRISM [33, 34, 23].

In this paper we focus on the qualitative model checking of finite state Markov decision processes (MDPs) against LTL properties. We propose a distributed-memory algorithm that solves the problem and exhibits the complexity of the sequential approach. This is a surprising result as the parallelization of LTL model checking usually costs extra time or space.

We use the automata-theoretic approach [29, 12, 14]. From the negation of a formula we construct a deterministic automaton on infinite words and check the existence of an accepting end component in the product-MDP resulting from the given MDP and the constructed automaton (probabilistic satisfaction problem).

It is very important to stress that this approach requires a determinization of the Büchi automaton obtained from the LTL formula. If the initial Büchi automaton is deterministic, the probabilistic emptiness problem can be solved in polynomial time [31]. As deterministic Büchi automata are strictly less powerful than nondeterministic, one has to go to a more general type of $\omega$-automaton. In [29] deterministic Rabin automata are used, while in [12] the authors consider Büchi automata deterministic in limit which leads to a slight improvment of the complexity of [29]. In the sequential case the LTL model checking problem for MDPs is hard for doubly exponential time, and can be solved in time doubly exponential in the specification and quadratic in the size of the program.

The sequential algorithms check the probabilistic satisfaction problem by repeated decomposition of the product graph into strongly connected components and subsequent removing of states that violate the "ergodic" condition. Distributed decomposition of a graph into SCCs is difficult to parallelize. Therefore, our new algorithm relies on a radically different approach for checking the probabilistic satisfaction problem. The basic idea comes from the distributed SCC-based algorithm for LTL model checking of non-probabilistic systems. To check the probabilistic satisfaction problem it is not necessary to decompose the graph into SCCs as the existence of an accepting end component can be checked easier by repeated reachability which systematically eliminates states that cannot belong to any accepting end components. Our algorithm as presented here has the complexity $\mathcal{O}(|M|^2 \cdot 2^{2^{\mathcal{O}(|\varphi|)}})$.

## 2 Qualitative LTL Model Checking

**Rabin automata.** A deterministic Rabin automaton is a tuple $A = (\Sigma, Q, q_{init}, \delta, Acc)$, where $\Sigma$ is a finite alphabet, $Q$ is a finite set of states, $q_{init} \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q$ is a (complete) transition function and $Acc = [(L_1, U_1), \ldots, (L_k, U_k)]$, with $L_i, U_i \subseteq Q$ for $i = 1, \ldots, k$, is an acceptance condition.

A run of $A$ over an infinite word $w = a_1 a_2 \ldots$ is a sequence $q_0, q_1, \ldots$, where $q_0 = q_{init}$ and $\delta(q_{i-1}, a_i) = q_i$ for all $i \geq 1$. Acceptance is defined in terms of limits. The limit of a run $r = q_0, q_1, \ldots$ is the set $\lim(r) = \{q \mid q = q_i \text{ infinitely often}\}$. A run $r$ is accepting if $\lim(r) \cap L_i \neq \emptyset$ and $\lim(r) \cap U_i = \emptyset$ for some $i$. We denote by $L(A)$ the set of all infinite words with an accepting run.

**Linear Temporal Logic (LTL).** Formulas of LTL are built from a set $AP$ of atomic propositions and are closed under the application of Boolean connectives, the unary temporal connective $X$ (next), and the binary temporal connective $U$ (until). LTL is interpreted over computations. A computation is a function $\pi : \omega \to AP$, which assigns truth values to the elements of $AP$ at each time instant and as such can be viewed as infinite words over the alphabet $2^{AP}$. For an LTL formula $\varphi$ we denote by $L(\varphi)$ the set of all computations satisfying $\varphi$.

**Proposition 2.1 ([32, 28]).** *Given an LTL formula $\varphi$, one can build a deterministic Rabin automaton $A$ with $2^{2^{\mathcal{O}(|\varphi|\cdot|\log\varphi|)}}$ states and $2^{\mathcal{O}(|\varphi|)}$ pairs in acceptance condition, such that* $L(A) = L(\varphi)$.

The transformation from LTL formulas to deterministic Rabin automata via nondeterministic Büchi automata [32] and Safra's [28] algorithm leads to a worst case double exponential blowup, which roughly meets the lower bound established in [21].

**Markov decision process (MDP).** We use MDP as a model of asynchronous probabilistic systems. In an MDP, any state $s$ might have several outgoing action-labeled transitions, each of them is associated with a probability distribution which yields the probabilities for the successor states. In addition, a labeling function attaches to any state $s$ a set of atomic propositions that are assumed to be fulfilled in state $s$. The atomic propositions will serve as atoms to formulate the desired properties in a temporal logic framework.

Formally, a Markov decision process [16, 26, 30] is a tuple $M = (S, \mathit{Act}, P, s_{init}, AP, L)$, where $S$ is a finite set of states, *Act* is a finite set of actions, $P : (S \times \mathit{Act} \times S) \to [0, 1]$ is a (three-dimensional) probability matrix, $s_{init} \in S$ is the initial state , $AP$ is a finite set of atomic propositions, and $L : S \to 2^{AP}$ is a labeling function. $\mathit{Act}(s)$ denotes the set of actions that are enabled in state $s$, i.e. the set of actions $\alpha \in \mathrm{Act}$ such that $P(s, \alpha, t) > 0$ for some state $t \in S$. For any state $s \in S$, we require that $\mathrm{Act}(s) \neq \emptyset$ and $\forall \alpha \in \mathrm{Act}(s). \sum_{s' \in S} P(s, \alpha, s') = 1$.

The intuitive operational semantics of an MDP is as follows. If $s$ is the current state then an action $\alpha \in \mathrm{Act}(s)$ is chosen nondeterministically and is executed leading to a state $t$ with probability $P(s, \alpha, t)$. We refer to $t$ as an $\alpha$-successor of $s$ if $P(s, \alpha, t) > 0$.

State $s$ is called *deterministic* if only one action is enabled in $s$. If all states of an MDP are deterministic, the MDP is called *Markov chain*.

An infinite path in an MDP is a sequence $\tau = s_0, \alpha_1, s_1, \alpha_2, \ldots \in (S \times Act)^\omega$ such that $\alpha_i \in Act(s_{i-1})$ and $P(s_{i-1}, \alpha_i, s_i) > 0$ for any $i \geq 1$. A trajectory of a path $\tau$ is the word $L(s_0), L(s_1), L(s_2), \ldots$ over the alphabet $2^{AP}$ obtained by the projection of $\tau$ to the state labels. Finite paths are finite prefixes of infinite paths that end in a state. We use the notation $last(\sigma)$ for the last state of a finite path $\sigma$.

A *scheduler* is a function which resolves the nondeterminism of MDP, and thus, it yields an exact probability measure on sets of paths of an MDP. We consider deterministic history dependent schedulers which are given by a function $D$ assigning an action $D(\sigma) \in Act(last(\sigma))$ to every finite path $\sigma$. Given a a scheduler $D$, the behavior of $M$ under $D$ can be formalized as a (possibly infinite state) Markov chain.

**Verifying LTL Specifications.** Let $AP$ be the alphabet of LTL specification $\varphi$. For an MDP $M$ and a scheduler $D$ the set of trajectories that satisfy the specification $\varphi$ is measurable [29]. We use $Pr_{M,D}(L(\varphi))$ to denote the probability that a trajectory of $M$ under $D$ satisfies the specification $\varphi$. We say that $M$ satisfies $\varphi$ if for all schedulers $D$, $Pr_{M,D}(L(\varphi)) = 1$.

Our distributed algorithm comes out from the automata-based approach to LTL model checking. As in the non-probabilistic case, the model is synchronized with the automaton corresponding to the negation of the formula. However, unlike the non-probabilistic case, deterministic automata have to be used instead of non-deterministic Büchi automata. Since we consider deterministic Rabin automata, the synchronization results in an MDP with Rabin acceptance condition in our case. The model checking problem is thus reduced to the non-emptiness problem for the product MDP.

Let $M = (S, Act, P, s_{init}, AP, L)$ be an MDP. Let $A = (2^{AP}, Q, q_{init}, \delta, [(L_1, U_1), \ldots, (L_k, U_k)])$ be a deterministic Rabin automaton. The *synchronized product* of $M$ and $A$ is an MDP $M \times A = (S \times Q, Act_{M\times A}, P_{M\times A}, init, AP, L_{M\times A})$ *with Rabin acceptance condition* $Acc_{M\times A} = [(S \times L_1, S \times U_1), \ldots, (S \times L_k, S \times U_k)]$, where $Act_{M\times A}((u, v)) = Act(u)$, $init = (s_{init}, q_{init})$, $L_{M\times A}((u, v)) = L(u)$, and

$$P_{M\times A}((s, p), \alpha, (t, q)) = \begin{cases} P(s, \alpha, t) & \text{if } \delta(p, L(s)) = q \\ 0 & \text{otherwise.} \end{cases}$$

Our algorithm rests upon a connection between stochastic properties of an MDP and its structure when viewed as a graph-like structure. This is exemplified by notions of *end components* and *accepting end components* [14, 12].

Let $M \times A$ be a product MDP with Rabin acceptance condition. Consider a directed labeled graph $G_{M\times A} = (S \times Q, \mathsf{init}, E)$ where $\mathsf{init}$ is an initial state of $G_{M\times A}$, $E \subseteq (S \times Q) \times \mathsf{Act} \times (S \times Q)$, and $E = \{(u, \alpha, v) \mid P_{M\times A}(u, \alpha, v) > 0\}$.

A subgraph $(V', E')$ of $G_{M\times A}$ forms a *strongly connected component* (SCC) if for any two vertices $u, v \in V'$ there is a path from $u$ to $v$ in $(V', E')$. SCC is *non-trivial* if it has at least one edge. SCC is *terminal* if there is no edge $(u, \alpha, v) \in E$ outgoing from SCC, i.e. such that $u \in V'$ and $v \notin V'$. Let $(V', E')$ be a subgraph of $G_{M\times A}$. A vertex $u \in V'$ is *closed in* $(V', E')$ if

- there is $\alpha \in \mathsf{Act}_{M\times A}(u)$ and $v \in V'$ such that $(u, \alpha, v) \in E'$
- if $(u, \alpha, v) \in E'$, then $(u, \alpha, w) \in E'$ for every $w \in V$ such that $(u, \alpha, w) \in E$.

A subgraph $(V', E')$ of $G_{M\times A}$ is *closed under the positive probabilistic transitions* (closed for short) if every state in $V'$ is closed in $(V', E')$.

An *end component* (EC) in $G_{M\times A}$ is a strongly connected component of $G_{M\times A}$ that is reachable from the initial state $\mathsf{init}$ and is closed under the positive probabilistic transitions. The end component $(V', E')$ is called *maximal* if there is no other end component of $G_{M\times A}$ containing all vertices and all edges from $(V', E')$. End component is called *terminal* if it is a terminal SCC.

End component $(V', E')$ is *accepting* (AEC) with respect to the Rabin acceptance condition if for some $i$, $1 \leq i \leq k$, we have $V' \cap (S \times L_i) \neq \emptyset$ and $V' \cap (S \times U_i) = \emptyset$. We refer to the index $i$ as a *valid index*.

**Proposition 2.2 ([30]).** *Let $M$ be an MDP and $\varphi$ an LTL property. Let $A$ be a deterministic Rabin automaton with $L(A) = L(\varphi)$. Then there exists a scheduler $D$ such that $\mathrm{Pr}_{M,D}(L(\varphi)) > 0$ if and only if there is an accepting end component in the graph $G_{M\times A}$.*

The qualitative LTL model checking of MDPs is thus reduced to the question whether the $G_{M\times A}$ for a given MDP with the Rabin acceptance condition contains an accepting end component.

A sequential algorithm for AEC detection is given in [30, 14] and for a similar problem in [12]. The idea is to decompose the given graph $G_{M\times A} = (V, \mathsf{init}, E)$ with the Rabin acceptance condition $\mathsf{Acc}_{M\times A}$ into strongly connected components, and to test every component for closure under positive probabilistic transitions and for its acceptance with respect to individual pairs $(L, U) \in Acc_{M\times A}$. If either of the two conditions is violated, the blamed states are removed from the graph and the component is again decomposed into SCCs. The graph contains an AEC if and only if the final decomposition is nonempty. The complexity of the algorithm is determined by the number of acceptance pairs in $Acc_{M\times A}$, the complexity of the SCC decomposition, and the number of repeated SCC decompositions till stabilization. The SCC decomposition can be performed with Tarjan's algorithm in time linear in the size of the graph, the number of SCC decompositions is bounded by the number of vertices. Hence, the complexity of the algorithm is $\mathcal{O}(|Acc_{M\times A}| \cdot n^2)$, where $n$ is the number of vertices.

## 3 Approximation Set Algorithm

In this section we present a new sequential algorithm, prove its correctness, and give a complexity bound. The distributed version of the algorithm is discussed in the next section.

If we follow the classification of SCC-detection algorithms as presented in [27, 17], then the above sketched sequential algorithm can be classified as an AEC-enumeration algorithm as it enumerates all accepting end components of a graph. Contrary to this, the presented (distributed) algorithm can be classified as an AEC-hull algorithm as it computes the set of states that contains all accepting end components. In particular, the algorithm maintains *approximation set* of states that may belong to an AEC. The algorithm repeatedly refines the approximation set by locating and removing states that cannot belong to an AEC, we call this a *pruning step*. The core of the algorithm are conditions determining the states to prune.

Formally, let $M \times A$ be a product MDP, $G_{M\times A} = (S \times Q, init, E)$ be its corresponding graph, and $Acc_{M\times A} = [(S \times L_1, S \times U_1), \ldots, (S \times L_k, S \times U_k)]$ be the Rabin acceptance condition. Without lost of generality we suppose that all vertices in $S \times Q$ are reachable from the vertex *init*. The algorithm tests each index $i$, $i = 1, \ldots, k$ whether it is valid or not. We henceforth assume a fixed index $i$ and denote the pair $(S \times L_i, S \times U_i)$ as $(L, U)$ and refer to the vertices from $L$ and $U$ as $L$-states and $U$-states, respectively.

An *approximation graph* is a subgraph $(AS, E_{AS})$ of the graph $G_{M\times A}$ such that $AS \cap U = \emptyset$ and $(AS, E_{AS})$ contains all accepting end components of $G_{M\times A}$. Our goal is to formulate criteria for eliminating vertices and edges from the approximation graph.

```
proc DETECT-AEC($((S \times Q, init, E), (L, U))$)
   $AS := S \times Q \setminus U$
   $E_{AS} := E$
   $to\text{-}eliminate := U$
   CLOSURE()
   $oldSize := 0$
   while $(|AS| \neq oldSize \ \wedge \ |AS| > 0)$ do
            $oldSize := |AS|$
            L-REACHABILITY()
            CLOSURE()
   od
   return($\|AS\| > 0$)
end


proc L-REACHABILITY()
   $can\text{-}reach\text{-}L := \emptyset;$
   $to\text{-}explore := AS \cap L$
   while $(to\text{-}explore \neq \emptyset)$ do
            pick and remove $q$ from $to\text{-}explore$
            foreach $(r, \alpha, q) \in E_{AS}$ do
               if $(r \notin can\text{-}reach\text{-}L)$
                  then $can\text{-}reach\text{-}L := can\text{-}reach\text{-}L \cup \{r\}$
                       $to\text{-}explore := to\text{-}explore \cup \{r\}$
               fi
            od
   od
   $to\text{-}eliminate := AS \setminus can\text{-}reach\text{-}L$
   $AS := AS \cap can\text{-}reach\text{-}L$
end


proc CLOSURE()
   while $(to\text{-}eliminate \neq \emptyset)$ do
            pick and remove $q$ from $to\text{-}eliminate$
            foreach $(q, \alpha, p) \in E_{AS}$ do
               $E_{AS} := E_{AS} \setminus \{(q, \alpha, p)\}$
            od
            foreach $(r, \alpha, q) \in E_{AS}$ do
               foreach $(r, \alpha, p)$ do
                  $E_{AS} := E_{AS} \setminus \{(r, \alpha, p)\}$
               od
               $Act_{M \times A}(r) := Act_{M \times A}(r) \setminus \{\alpha\}$
               if $(Act_{M \times A}(r) = \emptyset \ \wedge \ r \in AS)$
                  then $to\text{-}eliminate := to\text{-}eliminate \cup \{r\}$
                       $AS := AS \setminus \{r\}$
               fi
            od
   od
end
```

Figure 1: Sequential algorithm

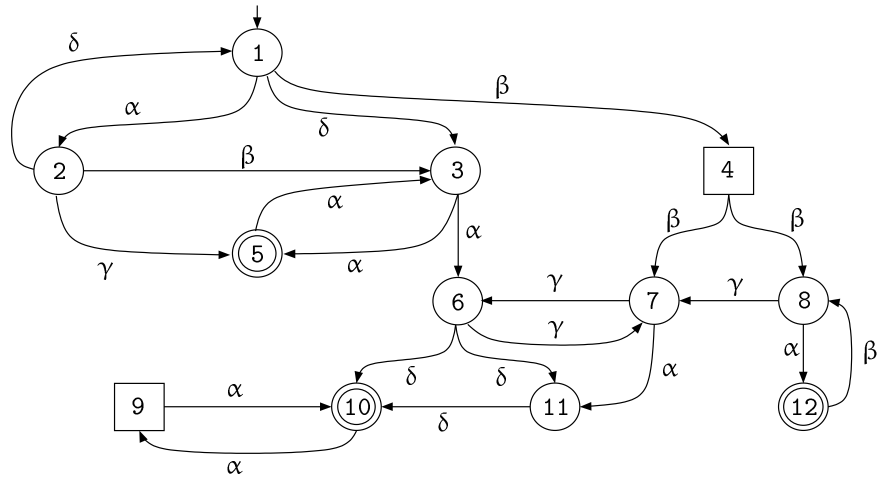

Figure 2: Example of a product MDP.

Let $(AS, E_{AS})$ be an approximation graph. Then the following two conditions are necessary for a vertex $v \in AS$ to belong to an AEC:

1. There is an L-state which is reachable from $v$ along a non-trivial path[1] in $(AS, E_{AS})$.
2. The vertex $v$ is *closed* in $(AS, E_{AS})$.

The first condition correspond to the acceptance condition for EC (here we remind that the approximation set does not contain U-states). The second condition conforms with the closeness under the positive probabilistic transitions.

**Lemma 3.1.** *Graph* $G_{M\times A} = (S \times Q, \mathtt{init}, E)$ *contains an AEC if and only if there is a non-empty approximation graph* $(AS, E_{AS})$ *such that all vertices from* $AS$ *meet the conditions 1 and 2.*

**Proof:** Any AEC in $G_{M\times A}$ is an approximation graph with vertices complying both conditions.

For the opposite case, let us assume that $(AS, E_{AS})$ is an approximation graph and all vertices in the set $AS$ meet the conditions 1 and 2. Let $X$ be a *terminal* SCC of $(AS, E_{AS})$. By the condition 1, $X$ contains at least one L-state and is nontrivial. From the condition 2 we have that $X$ is closed under positive probabilistic transitions. Altogether, $X$ is an accepting end component. $\square$

[1] A path in a graph is non-trivial if it contains at least one edge.

The pseudo-code of the algorithm DETECT-AEC is given in Figure 1. The algorithm starts with an approximation graph containing all vertices from $G_{M\times A}$ except U-states. In each iteration of the **while** loop, the vertices violating condition 1 are pruned in the procedure L-REACHABILITY while vertices violating the condition 2 are eliminated in the procedure CLOSURE. The iterations of the procedure DETECT-AEC are called *external*.

The computation of the procedure DETECT-AEC can be illustrated on the product MDP depicted in Figure 2. Vertices 5, 10, and 12 are L-states; vertices 4 and 9 are U-states. Initially, $AS = \{1, 2, 3, 5, 6, 7, 8, 10, 11, 12\}$ and *to-eliminate*$= \{4, 9\}$. First execution of CLOSURE removes from $E_{AS}$ the edges incident to vertices 4 and 9, i.e., $(4, \beta, 7), (4, \beta, 8), (1, \beta, 4), (9, \alpha, 10), (10, \alpha, 9)$. This causes that $Act_{M\times A}(10) = \emptyset$ and the vertex 10 is not closed in the current approximation graph. Therefore, the vertex is added to the set *to-eliminate* and removed from $AS$. Consequently, the edges $(6, \delta, 10), (6, \delta, 11)$, and $(11, \delta, 10)$ are removed from $E_{AS}$ as well. We have $Act_{M\times A}(11) = \emptyset$, the vertex 11 is added to *to-eliminate* and removed from $AS$. Then $(7, \alpha, 11)$ is removed from $E_{AS}$. As the set *to-eliminate* is now empty, the procedure CLOSURE terminates. The first external iteration is now started. The value of *oldSize* is 8. The procedure L-REACHABILITY detects vertices 6 and 7 as those from which none L-state is reachable and sets *to-eliminate* to $\{6, 7\}$ and *AS* to $\{1, 2, 3, 5, 8, 12\}$. Subsequent call to the procedure CLOSURE removes the edges $(7, \gamma, 6)$, $(8, \gamma, 7)$, $(6, \gamma, 7)$, and $(3, \alpha, 6), (3, \alpha, 5)$ from $E_{AS}$. $Act_{M\times A}(3)$ becomes empty, therefore, the vertex 3 is added to *to-eliminate* and removed from $AS$. Then the edges $(1, \delta, 3), (2, \beta, 3), (5, \alpha, 3)$ are removed from $E_{AS}$, the vertex 5 is added to *to-eliminate* and removed from $AS$. Finally, the edge $(2, \gamma, 5)$ is removed from $E_{AS}$. The set *to-eliminate* is now empty and the procedure CLOSURE terminates. In the second external iteration the value of *oldSize* is 4. The procedure L-REACHABILITY eliminates the vertices 1 and 2 and the procedure CLOSURE removes the edges $(1, \alpha, 2), (2, \delta, 1)$ from $E_{AS}$. There are no more vertices eliminated in the third external iteration. As the resulting approximation graph is non-empty, the procedure DETECT-AEC returns *true*.

In what follows we prove the correctness of the algorithm and analyze its complexity. $(AS, E_{AS})$ denotes the approximation graph at the beginning of an external iteration and $(\overline{AS}, \overline{E}_{AS})$ denotes the graph after the iteration has finished. $|G_{M\times A}|$ denotes the size of the product graph $G_{M\times A}$.

**Lemma 3.2.** *Upon termination of the procedure* CLOSURE, $\overline{E}_{AS}$ *contains only edges incident to vertices from* $\overline{AS}$ *and all vertices in* $\overline{AS}$ *are closed in* $(\overline{AS}, \overline{E}_{AS})$. *The complexity of* CLOSURE *is* $\mathcal{O}(|G_{M\times A}|)$.

**Proof:** Let the set *to-eliminate* contains all vertices directly violating the closure property in $(\overline{AS}, \overline{E}_{AS})$. Every iteration of the procedure maintains this property, which can be easily seen from the pseudo-code. Each vertex in *to-eliminate* is eventually removed and never inserted into the set again. As soon as the set is empty, the procedure terminates. □

**Lemma 3.3.** *Upon termination of the procedure* L-REACHABILITY, $\overline{AS} \subseteq AS$ *and* $\overline{AS}$ *contains only those vertices from which an* L*-state is reachable in* $(AS, E_{AS})$. *The complexity of* L-REACHABILITY *is* $\mathcal{O}(|G_{M\times A}|)$.

**Proof:** The procedure adds vertices to the set *can-reach-L* only when an edge leading to an L-state, or leading to a vertex from *can-reach-L*, is discovered. No vertex lying outside of AS can be added to the set *can-reach-L* as only edges in $E_{AS}$ are explored and $(AS, E_{AS})$ is closed. This is due to Lemma 3.2 and the fact that each call to L-REACHABILITY is preceded by a call to CLOSURE.

The complexity is given by the fact that every edge in $E_{AS}$ is explored at most once. The procedure proceeds by backward search as this minimizes its complexity. If a forward search was employed the complexity would be $|AS|$ times the complexity of the forward search. □

**Lemma 3.4.** *Upon termination of the procedure* DETECT-AEC *every vertex in the approximation set* AS *meets the conditions 1 and 2.*

**Proof:** The procedure DETECT-AEC removes all U-states from V and applies CLOSURE to ensure that all vertices in AS are closed (Lemma 3.2). Due to Lemmas 3.2 and 3.3 only vertices violating either 1 or 2 are removed from the approximation set in each external iteration. Once the external iteration does not change the approximation set all vertices in AS meet both conditions. □

To prove the complexity bound we need to give an upper bound on the number of external iterations. A trivial upper bound is $|S \times Q|$ as in each external iteration the size of AS is decreased. However, a more precise bound can be given in terms of maximal end components. At the very beginning, the algorithm DETECT-AEC removes from the graph all $U$-states and some edges to guarantee that all remaining vertices are closed. We refer to this graph as $G_{M\times A} \setminus U$.

**Lemma 3.5.** *The number of external iterations of* DETECT-AEC *is no more than the number of maximal end components in* $G_{M\times A} \setminus U$*.*

**Proof:** The key observation is that in each external iteration of DETECT-AEC either all vertices from at least one maximal end component in $G_{M\times A} \setminus U$ are removed from the approximation set, or the approximation set is not changed at all (and the computation of DETECT-AEC finishes). Furthermore, at the beginning of each external iteration the approximation graph is closed (Lemma 3.2).

Let $C = (V_C, E_C)$ be a maximal EC in $G_{M\times A} \setminus U$. Let us suppose that in some external iteration a vertex $q \in V_C$ is removed from the approximation set. If the vertex $q$ is removed by the procedure L-REACHABILITY because none $L$-state is reachable from $q$, then together with $q$ the whole $V_C$ is removed from the approximation set (EC is strongly connected). If no vertex is removed by L-REACHABILITY, then no vertex can be removed by CLOSURE as the approximation graph is closed and the algorithm would terminate.

□

**Theorem 3.1.** *Let* $M$ *be an MDP and* $\varphi$ *be an LTL formula. Then the question whether for all schedulers* $D$*,* $\Pr_{M,D}(L(\varphi)) = 1$*, can be correctly solved by the* DETECT-AEC *algorithm in time* $\mathcal{O}(|M|^2 \cdot 2^{2^{|\varphi|\cdot\log|\varphi|}})$*.*

**Proof:** Lemma 3.4 together with Lemma 3.1 give correctness of the DETECT-AEC algorithm.

If we start with an MDP $M$ that has $m$ states and $e$ transitions, then $G_{M\times A}$ has no more than $m \cdot 2^{2^{|\varphi|\cdot\log|\varphi|}}$ vertices and $e \cdot 2^{2^{|\varphi|\cdot\log|\varphi|}}$ edges, i.e. the size of the product graph $G_{M\times A}$ is $\mathcal{O}(|M| \cdot 2^{2^{|\varphi|\cdot\log|\varphi|}})$. Complexity of DETECT-AEC is in the worst case quadratic in the size of the product graph (though Lemma 3.5 gives a more precise bound). The algorithm is performed for every acceptance pair in the corresponding Rabin automaton.
□

Courcoubetis and Yannakakis [12] give an algorithm for qualitative LTL model checking of MDP with somewhat better complexity $\mathcal{O}(|M|^2 \cdot 2^{2^{\mathcal{O}(|\varphi|)}})$. This is due to the fact that their algorithm translates the verified property to a Büchi automaton which is deterministic in limit. However, our algorithm is based on a translation to a deterministic Rabin automaton. The approach we present is independent of the type of the

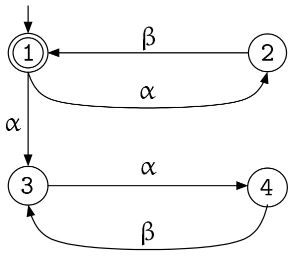

Figure 3: Modification of Condition 1.

$\omega$-automaton. Therefore, using Büchi automata that are deterministic in limit our algorithm exhibits the same asymptotic complexity $\mathcal{O}(|M|^2 \cdot 2^{2^{\mathcal{O}(|\varphi|)}})$.

Our algorithm stores edges to enumerate predecessors. A natural question is, whether this is really necessary. While the condition 1 can be replaced by a symmetric condition requiring that the vertex $u$ is reachable from an $L$-state along a non-trivial path in $(AS, E_{AS})$ and tested by a forward reachability without using backward edges, the symmetric approach does not work in the case of the condition 2. This is illustrated on the graph in Figure 3 where all vertices are reachable from the $L$-state (its number is 1) and are closed, but the graph does not contain any AEC.

If an MDP contains deterministic states only (the MDP is a Markov chain), then every end component of the corresponding graph $G_{M\times A} \setminus U$ is a terminal one. As argued in the proof of Theorem 3.5 every terminal SCC is removed completely in an external iteration or it remains in the approximation graph forever. Therefore, the DETECT-AEC algorithm terminates on Markov chains after one iteration and its complexity is linear with respect to the size of the product graph.

## 4 Distributed Implementation of The Algorithm

In the distributed setting, such as the network of workstations, the graph to be explored is partitioned among the workstations using the so called partition function so that every single workstation is responsible for the subgraph assigned to it. For the principle of partitioning see e.g. [6, 24]. As workstations work concurrently and communicate by means of message passing, parallelism is introduced in the computation.

The graphs to be explored are given implicitly by the description of the initial vertex and a set of rules specifying how for a given vertex all of its immediate successors can be generated. In practical terms, we are thus able to compute immediate successors of

```
while $(\neg \mathit{Finished})$ do
    PROCESS-INCOMING-MESSAGES()
    if $(\mathit{to\text{-}explore} \neq \emptyset)$
      then pick and remove $q$ from $\mathit{to\text{-}explore}$
           foreach $(r, \alpha, q) \in E_{AS}$ do
             if (PARTITION$(r)$ is local )
               then if $(r \notin \mathit{can\text{-}reach\text{-}L})$
                      then $\mathit{can\text{-}reach\text{-}L} := \mathit{can\text{-}reach\text{-}L} \cup \{r\}$
                           $\mathit{to\text{-}explore} := \mathit{to\text{-}explore} \cup \{r\}$
                    fi
               else send $r$ to $\mathit{can\text{-}reach\text{-}L}$ and $\mathit{to\text{-}explore}$
                    on PARTITION$(r)$
             fi
           od
    fi
od
```

Figure 4: Main loop of the distributed procedure L-REACHABILITY.

a given vertex, but we are not able to directly enumerate its predecessors. As our algorithm requires predecessors, vertices of the graph have to be generated first and all the edges stored. In particular, every vertex has an associated list of (pointers to) its immediate predecessors allowing thus every single workstation to enumerate successors as well as predecessors of vertices it is responsible for.

The implementation of the algorithm requires also a few other values to be stored at each vertex. In particular, these are the bit to distinguish whether the vertex belongs to the approximation set $AS$ and list of actions $Act_{M \times A}$ whose corresponding edges are still considered to be a part of the approximation set. The global sets $\mathit{to\text{-}eliminate}$, $\mathit{can\text{-}reach\text{-}L}$, and $\mathit{to\text{-}explore}$ are partitioned using the same partition function as the graph. If a vertex is about to be inserted into one of these sets, it is at first judged by the partition function and then sent to the workstation owning the vertex in order to be inserted in the corresponding local part of the set.

The main loop of the procedure L-REACHABILITY is replaced with the pseudo-code given in Figure 4 in the case of the distributed algorithm. The loop in the new pseudo-code terminates when all sent messages have been delivered and all local sets $\mathit{to\text{-}explore}$ are empty, which is detected using the standard distributed termination detection procedure and indicated with the flag *Finished*.

Procedure CLOSURE is modified following the same scheme. A specific problem arises when all edges with a given action are about to be removed from the set $E_{AS}$ for an immediate predecessors of a given vertex. Consider the situation as depicted on the left hand side of Figure 5. Let vertex 1 be *picked and removed* from the set $\mathit{to\text{-}eliminate}$

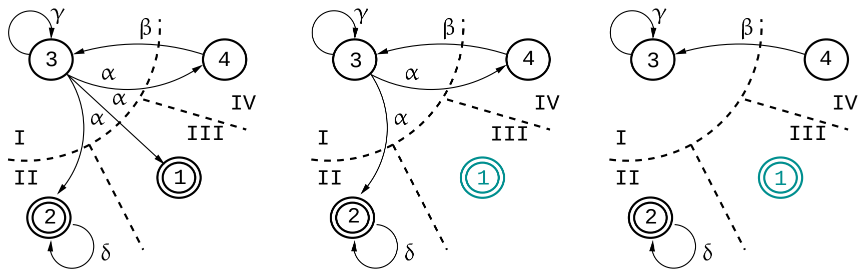

Figure 5: Distributed closure computation on approximation sets.

on the workstation III. Since it has no outgoing edges ($Act_{M\times A}(1)$ is empty) the list of immediate predecessors associated with the vertex is cleared, and the immediate predecessors are told to remove the corresponding action from their sets of valid actions. The only immediate predecessor of the vertex 1 is the vertex 3 that is assigned to the workstation I. Thus, a message requesting removal of the action $\alpha$ from the vertex 3 is sent from the workstation III to the workstation I and the vertex 1 is removed from the set $AS$ of vertices remaining in the approximation set (the bit representing its presence in the set is set to false). This is exemplified in the middle of the Figure. Once the workstation I receives the message it appropriately modifies the set $Act_{M\times A}(3)$, and sends messages to the workstations II and IV responsible for vertices 2 and 4, respectively, in order to update the corresponding lists of immediate predecessors of these vertices. As soon as this is done, the computation of the closure procedure for the vertex 1 is complete (see the situation on the right hand side of Figure 5).

## 5 Conclusions

We addressed the problem of qualitative verification of finite state Markov decision processes with respect to specifications expressed in linear temporal logic LTL. An optimal sequential algorithm for the problem is given in [12]. This algorithm is based on the decomposition of a graph into strongly connected components and as such cannot be directly modified and effectively implemented in a distributed setting.

We provide a new algorithm for qualitative LTL model checking of Markov decision processes with the same asymptotic complexity as given in [12]. Contrary to this algorithm, our algorithm does not require the decomposition into strongly connected components. Instead of this, the core operation of our algorithm is a reachability test.

Therefore, the algorithm can be easily implemented as a distributed-memory algorithm while preserving its complexity.

## References

[1] A. Aziz, V. Singhal, and F. Balarin. It usually works: The temporal logic of stochastic systems. In *Computer Aided Verification, 7th International Conference*, volume 939 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 1995.

[2] C. Baier. On the algorithmic verification of probabilistic systems. Habilitation Thesis, Universität Mannheim, 1998.

[3] C. Baier, F. Ciesinski, and M. Gröeßer. Quantitative analysis of distributed randomized protocols. In *FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems*. ACM Press, 2005.

[4] C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *Automata, Languages and Programming, 24th International Colloquium (ICALP'97)*, volume 1256 of *Lecture Notes in Computer Science*, pages 430–440. Springer, 1997.

[5] C. Baier, M. Größer, and F. Ciesinski. Partial order reduction for probabilistic systems. In *1st International Conference on Quantitative Evaluation of Systems (QEST 2004)*, pages 230–239. IEEE Computer Society, 2004.

[6] Jiří Barnat. *Distributed Memory LTL Model Checking*. PhD thesis, Faculty of Informatics, Masaryk University Brno, 2004.

[7] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings of FST&TCS 1995*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[8] B.Jeannet, P.dŠArgenio, and K.G. Lar. RAPTURE: A tool for verifying Markov Decision Processes. In *Proc. Tools Day / CONCURŠ02. Tech.Rep. FIMU-RS-2002-05*, pages 84–98. MU Brno, 2002.

[9] F. Ciesinski and C. Baier. Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. submitted for publication, 2006.

[10] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[11] C. Courcoubetis and M. Yannakakis. Verifying Temporal Properties of Finite-State Probabilistic Programs. In *29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 338–345. IEEE Computer Society Press, 1988.

[12] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[13] J-M. Couvreur, N. Saheb, and G. Sutre. An Optimal Automata Approach to LTL Model Checking of Probabilistic Systems. In *Logic for Programming, Artificial Intelligence, and Reasoning, 10th International Conference (LPAR 2003)*, volume 2850 of *Lecture Notes in Computer Science*, pages 361–375. Springer, 2003.

[14] L. de Alfaro. *Formal Verification of Stochastic Systems*. PhD thesis, Stanford University, Department of Computer Science, 1997.

[15] Luca de Alfaro. Stochastic transition systems. In *CONCUR '98: Concurrency Theory, 9th International Conference*, volume 1466 of *Lecture Notes in Computer Science*. Springer, 1998.

[16] C. Derman. *Finite-State Markovian Decision Processes*. Academic Press, New York, 1970.

[17] K. Fisler, R. Fraer, G. Kamhi, Y. Vardi, and Z. Yang. Is there a best symbolic cycle-detection algorithm? In *Proc. TACAS 2001*, volume 2031 of *LNCS*, pages 420–434. Springer, 2001.

[18] S. Hart and M. Sharir. Probabilistic Temporal Logics for Finite and Bounded Models. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–13. ACM, 1984.

[19] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[20] M. Huth and M. Kwiatkowska. Quantitative analysis and model checking. In *Proceedings of LICS'97*, pages 111–122. IEEE Computer Society Press, 1997.

[21] O. Kupferman and M. Vardi. Freedom, weakness, and determinism: From linear-time to branching-time. In *Proc. LICS 1998*, pages 81–92. IEEE Computer Society Press, 1998.

[22] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. *STTT*, 6(2):128–142, 2004.

[23] M. Kwiatkowska, D. Parker, Y. Zhang, and R. Mehmood. Dual-processor parallelisation of symbolic probabilistic model checking. In *Proc. 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'04)*, pages 123–130. IEEE Computer Society Press, 2004.

[24] Flavio Lerda and Riccardo Sisto. Distributed-memory Model Checking with SPIN. In *Proc. of the 5th International SPIN Workshop*, volume 1680 of *LNCS*. Springer-Verlag, 1999.

[25] C. Löding. Optimal bounds for the transformation of omega-automata. In *Proc. FST & TCS 1999*, volume 1738 of *LNCS*, pages 97–109. Springer, 1999.

[26] M. L. Puterman. *Markov Decision Processes-Discrete Stochastic Dynamic Programming*. John Wiley &Sons, New York, 1994.

[27] K. Ravi, R. Bloem, and F. Somenzi. A comparative study of symbolic algorithms for the computation of fair cycles. In *Proc. FMCAD 2000*, volume 1954 of *LNCS*, pages 143–160. Springer, 2000.

[28] S. Safra. On the complexity of $\omega$-automata. In *Proc. FOCS 1988*, pages 319–327. IEEE Computer Society Press, 1988.

[29] M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. FOCS 1985*, pages 327–338. IEEE Computer Society Press, 1985.

[30] M. Vardi. Probabilistic linear-time model checking: an overview of the automata-theoretic approach. In *Proc. Formal Methods for Real-Time and Probabilistic Systems, ARTS 1999*, volume 1601 of *LNCS*, pages 265–276. Springer, 1999.

[31] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings, Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Computer Society, 1986.

[32] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[33] Y. Zhang, D. Parker, and M. Kwiatkowska. Grid-enabled probabilistic model checking with PRISM. In *Proc. 4th All Hands Meeting (AHM'05)*, 2005.

[34] Y. Zhang, D. Parker, and M. Kwiatkowska. A wavefront parallelisation of CTMC solution using MTBDDs. In *Proc. International Conference on Dependable Systems and Networks (DSN'05)*, pages 732–742. IEEE Computer Society Press, 2005.