

Lazy Determinisation for Quantitative Model Checking^{*}

Ernst Moritz Hahn¹, Guangyuan Li¹, Sven Schewe², and Lijun Zhang¹

¹ State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences

² University of Liverpool

Abstract. The bottleneck in the quantitative analysis of Markov chains and Markov decision processes against specifications given in LTL or as some form of nondeterministic Büchi automata is the inclusion of a determinisation step of the automaton under consideration. Such determinisation constructions are notoriously difficult and cumbersome to implement. In this paper we argue that applying this step in full generality is only required in some places, and can often be circumvented by subset and breakpoint constructions. We have implemented our approach in a prototype tool. Our experiments show that our prototype can compete with mature tools like PRISM.

1 Introduction

Markov chains (MC) and Markov decision processes (MDP) are widely used to model and analyse systems that exhibit probabilistic and both probabilistic and nondeterministic choices, respectively. Properties of these systems are often specified by temporal logical formulas, such as the branching time logic PCTL [HJ94], the linear time logic PLTL, or their combination PCTL* [BdA95]. Model checking algorithms for PCTL can be solved in polynomial time [BdA95], but the model checking algorithm for linear time properties specified in PLTL is more expensive: for MCs the problem is in PSPACE, and for MDPs the problem is complete for double exponential time [CY95].

In classical model checking, to check whether a model $\mathcal{M}$ satisfies a property specified as an LTL formula φ , first a nondeterministic Büchi automaton $\mathcal{B}_{\neg\varphi}$ is constructed [VW86], which recognises the models of its negation $\neg\varphi$. Then, the model checking problem reduces to an emptiness check for the product $\mathcal{M} \times \mathcal{B}_{\neg\varphi}$. The translation to Büchi automata takes exponential time in the size of the formula, and one can apply standard off-the-shelf tools like LTL3BA [BKRS12] or SPOT [DL11] for it. This translation is mostly very efficient in practice due to the fact that these tools are highly optimised.

The quantitative analysis of a probabilistic model $\mathcal{M}$ against an LTL specifications φ is more involved. To compute the maximal probability $\text{Prob}^{\mathcal{M}}(\varphi)$ that φ is satisfied in $\mathcal{M}$, the automata-based approach includes the determinisation of an intermediate Büchi automaton $\mathcal{B}_{\varphi}$. If such a deterministic automata $\mathcal{A}$ is constructed for $\mathcal{B}_{\varphi}$, then determining the probability $\text{Prob}^{\mathcal{M}}(\varphi)$ reduces to solving an equation system for Markov chains, and a linear programming problem for MDPs [BdA95], both in the product $\mathcal{M} \times \mathcal{A}$. Such

^{*} Supported by Chinese Academy of Sciences fellowship for young international scientists (Grant No. 2013Y1GB0006), by the Engineering and Physical Science Research Council (EPSRC) through the grant EP/H046623/1 “Synthesis and Verification in Markov Game Structures”, and by NSFC-61061130541.

a determinisation step usually exploits a variant of Safra’s [Saf88] efficient determinisation construction, such as the techniques presented in [Pit07,Sch09b]. However, the complexity of these translations is $n^{O(n)}$, and such determinisation constructions are notoriously difficult and cumbersome to implement. Thus, model checkers supporting LTL, such as LIQUOR [CB06] or PRISM [KNP11] might fail to verify certain complex properties.

In this paper we argue that applying this determinisation step in full generality is only required in some places, while simpler subset and breakpoint constructions suffice elsewhere. A subset construction is known to be sufficient to determinise finite automata, but it fails for Büchi automata. Our first idea is to construct under- and over-approximation starting from the subset construction. That is, we construct two (deterministic) subset automata $\mathcal{S}^u$ and $\mathcal{S}^o$ such that $\mathcal{L}(\mathcal{S}^u) \subseteq \mathcal{L}(\mathcal{B}_\varphi) \subseteq \mathcal{L}(\mathcal{S}^o)$ where $\mathcal{L}(\mathcal{B}_\varphi)$ denotes the language defined by the LTL specification φ . The subset automata $\mathcal{S}^u$ and $\mathcal{S}^o$ are the same except for their accepting conditions. We build a product Markov chain with the subset automata. We establish a nice property that the probability $\text{Prob}^{\mathcal{M}}(\varphi)$ equals the probability of reaching some *accepting* leaf strongly connected components (SCC) in the product. For each leaf SCC S in the product, we can first use the accepting conditions in $\mathcal{S}^u$ or $\mathcal{S}^o$ to determine whether S is accepting or rejecting, respectively. The challenge remains when the test is inconclusive. In this case, we refine S using a breakpoint construction. Finally the full Rabin determinisation step is used if the breakpoint construction fails as well. We discuss that the framework can be easily adapted to MDPs by analysing the end components [CY95,BdA95].

We have implemented our approach in a prototype tool, which we applied on various Markov chain and MDP case studies. Our experiments show that our prototype can compete with mature tools like PRISM.

Related Work. The first paper about automata-based LTL model checking [VW86] already discussed probabilistic systems, but only their qualitative analysis. For an overview of model checking (including LTL and probabilistic models) see [BK08]. LIQUOR [CB06] is an LTL model checker for MDPs that uses the transformation of LTL formulas to deterministic Rabin or Streett automata. PROBDIVINE-MC [BBC⁺08] follows a similar approach but is particularly tuned for parallel computing. The well-known probabilistic model checker PRISM [KNP11] also works similarly. A recent publication [CGK13] proposed an improved algorithm for model checking a restricted version of LTL by using an extension of Rabin automata. The transformation of LTL into this extended class of deterministic automata allows for the improved bound of $2^{O(2^n)}$ where n is the size of the formula, while the total complexity of the transformation from LTL to deterministic Rabin automata is in $2^{O(2^{n+\log_2 n})}$ [KE12].

Finally we remark that for Markov chains there are efficient algorithms. The paper [CY95] presents a single exponential algorithm for LTL specifications without using automata theory. A recent paper [BLW13] shows that unambiguous Büchi automata can be used for Markov chains, which gives also a single exponential algorithm. On the other side, model checking MDPs against LTL is known to be complete for double exponential time [CY95]. The two approaches above cannot be extended to MDPs, whereas our approach in this paper applies for MCs and MDPs.

Outline. The paper is organised as follows. We briefly review some definitions in Section 2. The main contribution of this paper, the lazy approximation construction, is pre-

sented for Markov chains in Section 3 and extended to MDPs in Section 4. Experimental results are presented in Section 5.

2 Preliminaries

2.1 Nondeterministic and Deterministic Automata

Nondeterministic Rabin automata are used to represent ω -regular languages $L \subseteq \Sigma^\omega = \omega \rightarrow \Sigma$ over a finite alphabet Σ . In this paper, we use automata with trace-based acceptance mechanisms. We denote the set $\{1, 2, \dots, k\}$ by $[1..k]$.

A (transition-labelled) nondeterministic Rabin automaton with k accepting pairs is a quintuple $\mathcal{A} = (\Sigma, Q, I, T, \{(A_i, R_i) \mid i \in [1..k]\})$, consisting of a finite alphabet Σ , a finite set Q of states with a non-empty subset $I \subseteq Q$ of initial states, a set $T \subseteq Q \times \Sigma \times Q$ of transitions from states through input letters to successor states, and a finite family $\{(A_i, R_i) \in 2^T \times 2^T \mid i \in [1..k]\}$ of Rabin pairs. (For convenience, we sometimes use other finite sets of indices than $[1..k]$.)

Nondeterministic Rabin automata are interpreted over infinite sequences $\alpha: \omega \rightarrow \Sigma$ of input letters. An infinite sequence $\rho: \omega \rightarrow Q$ of states of $\mathcal{A}$ is called a *run* of $\mathcal{A}$ on an input word α if the first letter $\rho(0) \in I$ of ρ is an initial state, and if, for all $i \in \omega$, $(\rho(i), \alpha(i), \rho(i+1)) \in T$ is a transition. For a run ρ of a word α , we denote with $\bar{\rho}: i \mapsto (\rho(i), \alpha(i), \rho(i+1))$ the *transitions* of ρ .

A run $\rho: \omega \rightarrow Q$ is *accepting* if, for some index $i \in [1..k]$, some transition $t \in A_i$ in the accepting set A_i of the Rabin pair (A_i, R_i) , but no transition $t' \in R_i$ from the rejecting set R_i of this Rabin pair appears infinitely often in the transitions $\bar{\rho}$ of ρ . ($\exists i \in [1..k]. \inf(\bar{\rho}) \cap A_i \neq \emptyset \wedge \inf(\bar{\rho}) \cap R_i = \emptyset$ for $\inf(\bar{\rho}) = \{t \in T \mid \forall i \in \omega \exists j > i \text{ such that } \bar{\rho}(j) = t\}$). A word $\alpha: \omega \rightarrow \Sigma$ is *accepted* by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run on α , and the set $\mathcal{L}(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \alpha \text{ is accepted by } \mathcal{A}\}$ of words accepted by $\mathcal{A}$ is called its *language*.

For technical convenience we also allow for finite runs $q_0 q_1 q_2 \dots q_n$ with $T \cap \{q_n\} \times \{\alpha(n)\} \times Q = \emptyset$. Stated differently, no actions are enabled from q_n . Naturally, no finite run satisfies the Rabin condition. Finite runs are rejecting, and have no influence on the language of an automaton.

Transition-labelled generalised Büchi automata (TGBA) $\mathcal{B} = (\Sigma, Q, I, T, \{F_i \mid i \in [1..k]\})$ have a family of accepting (or: final) sets. A run ρ of a TGBA is accepting if it contains infinitely many transitions from all final sets ($\forall i \in [1..k]. \inf(\bar{\rho}) \cap F_i \neq \emptyset$).

Let $\delta: (q, \sigma) \mapsto \{q' \in Q \mid (q, \sigma, q') \in T\}$ denote the successor function for a set of transitions. A Rabin or TGBA is called *deterministic* if δ is deterministic ($\forall (q, \sigma) \in Q \times \Sigma. |\delta(q, \sigma)| \leq 1$) and $I = \{q_0\}$ is singleton.

For notational convenience, we denote deterministic automata by the tuple $(\Sigma, Q, q_0, \delta, \mathcal{ACC})$, where $\mathcal{ACC}$ is the acceptance condition (Rabin or TGBA), and $\delta: Q \times \Sigma \rightarrow Q$ is the partial function, which is defined at (q, σ) iff σ is enabled at q .

2.2 Determinisation of Generalised Büchi Automata

Generalised Büchi automata can be translated to deterministic Rabin automata (DRAs) using the following construction from [SV12]. We first define the structure that captures the acceptance mechanism of our deterministic Rabin automaton.

Ordered trees. We call a tree $\mathcal{T} \subseteq \mathbb{N}^*$ an *ordered tree* if it satisfies the following constraints, and use the following terms:

- every element $v \in \mathcal{T}$ is called a *node*,
- the empty sequence ε , called the *root*, is in $\mathcal{T}$,
- if a node $v = n_1 \dots n_j n_{j+1}$ is in $\mathcal{T}$, then $v' = n_1 \dots n_j$ is also in $\mathcal{T}$,
 v' is called the *predecessor* of v , denoted $p(v)$ ($p(\varepsilon)$ is undefined),
- if a node $v = n_1 \dots n_j$ is in $\mathcal{T}$, then $v' = n_1 \dots n_{j-1}i$ is also in $\mathcal{T}$ for $0 < i < n_j$,
 v' is called an *older sibling* of v , the set of older siblings of v is denoted $\text{os}(v)$.

Ordered trees are therefore non-empty sets of nodes that are closed under predecessors and older siblings.

Generalised history tree. A generalised history tree (GHT) d over Q for k accepting sets is a triple $d = (\mathcal{T}, l, h)$ such that:

- $\mathcal{T}$ is an ordered tree,
- $l: \mathcal{T} \rightarrow 2^Q \setminus \{\emptyset\}$ is a labelling function such that
 - $l(v) \subsetneq l(p(v))$ holds for all $\varepsilon \neq v \in \mathcal{T}$,
 - the intersection of the labels of two siblings is disjoint
 $(\forall v \in \mathcal{T}. \forall v' \in \text{os}(v). l(v) \cap l(v') = \emptyset)$, and
 - the union of the labels of all siblings is *strictly* contained in the label of their predecessor
 $(\forall v \in \mathcal{T} \exists q \in l(v) \forall v' \in \mathcal{T}. v = p(v') \Rightarrow q \notin l(v'))$,
- $h: \mathcal{T} \rightarrow [1..k]$ is a function that labels every node with a natural number from $[1..k]$.

For a GHT $d = (\mathcal{T}, l, h)$, $(\mathcal{T}, l)$ is the history tree introduced in [Sch09b]. GHTs are enriched by the second labelling function, h , which is used to relate nodes with a particular accepting set. The construction separates the transition mechanism from the acceptance condition.

A GHT contains the set of reachable states in its root. We therefore denote $l(\varepsilon)$ for a given GHT $d = (\mathcal{T}, l, h)$ by $\text{reach}(d)$.

Determinisation construction [SV12]. Let $\mathcal{B} = (\Sigma, Q, I, T, \{F_i \mid i \in [1..k]\})$ be a TGBA with $|Q| = n$ states and k accepting sets. We will construct an equivalent deterministic Rabin automaton $\mathcal{D} = (\Sigma, D, d_0, \delta, \{(A_i, R_i) \mid i \in J\})$, denoted $\text{det}(\mathcal{B})$, as follows.

- D is the set of generalised history trees over Q .
 - d_0 is the generalised history tree $(\{\varepsilon\}, l: \varepsilon \mapsto I, h: \varepsilon \mapsto 1)$.
 - For every tree $d \in D$ and letter $\sigma \in \Sigma$, the transition $d' = \delta(d, \sigma)$ is the result of the following sequence of transformations:
 1. *Raw update of l .* For every node $v \in d$ with label Q' , replace the label of v with the σ successors $\{q \in Q \mid \exists q' \in Q'. (q', \sigma, q) \in T\}$ of Q' .
 2. *Sprouting new children.* For every node $v \in d$ with label Q' and c children such that $h(v) = i$, create a new child vc with $l(vc) = \{q \in Q \mid \exists q' \in Q'. (q', \sigma, q) \in F_i\}$ and $h(vc) = 1$.
 3. *Stealing of labels.* For every node v with label $l(v) = Q'$ and all states $q \in Q'$, q is removed from the labels of all younger siblings of v and all of their descendants.
 4. *Accepting and removing.* For every node v whose label is equal to the union of the labels of its children, remove all descendants of v from the tree, and restrict the domain of l and h accordingly. If $h(v) = k$, change $h(v)$ to 1 and otherwise increment $h(v)$ by one.
- The transition is in A_v for all nodes v for which this applies.

5. *Removing nodes.* Remove all nodes with empty labels. (The resulting tree $\mathcal{T}'$ may no longer be ordered.) l and h are updated by restricting their domain.
6. *Renaming and rejecting.* To repair the orderedness, we call $\|v\| = |\text{os}(v) \cap \mathcal{T}'|$ the number of (still existing) older siblings of v , and map $v = n_1 \dots n_j$ to $v' = \|n_1\| \|n_1 n_2\| \|n_1 n_2 n_3\| \dots \|v\|$, denoted $\text{rename}(v)$.
We update a triple $(\mathcal{T}, l, h)$ from the previous step to $(\text{rename}(\mathcal{T}), l': \text{rename}(v) \mapsto l(v), h': \text{rename}(v) \mapsto h(v))$.
For all v not in $\mathcal{T}'$ and all nodes $v \in \mathcal{T}'$ with $\text{rename}(v) \neq v$, the transition is in R_v .

– J is the set of nodes that occur in some ordered tree of size n .

2.3 Markov Chains and Product

A distribution μ over M is a function $\mu: M \rightarrow [0, 1]$ such that $\sum_{m \in M} \mu(m) = 1$. A Markov chain (MC) is a tuple $\mathcal{M} = (M, L, \mu_0, P)$ where M is a finite set of states, $L: M \rightarrow \Sigma$ is a labelling function, μ_0 is the initial distribution, and $P: M \times M \rightarrow [0, 1]$ is a probabilistic transition matrix satisfying $\sum_{m' \in M} P(m, m') \in \{0, 1\}$ for all $m \in M$. A state m is called absorbing if $\sum_{m' \in M} P(m, m') = 0$.

A maximal path of $\mathcal{M}$ is an infinite sequence $\xi = m_0 m_1 \dots$ satisfying $P(m_i, m_{i+1}) > 0$ for all $i \geq 0$, or a finite one if the last state is absorbing. We denote by $\text{Paths}^{\mathcal{M}}$ the set of all maximal paths of $\mathcal{M}$. An infinite path $\xi = m_0 m_1 \dots$ defines the word $w_0 w_1 \dots \in \Sigma^\omega$ with $w_i = L(m_i)$.

For a finite sequence $m_0 m_1 \dots m_k$, we denote by $\text{Cyl}(m_0 m_1 \dots m_k)$ the set of maximal paths starting with prefix $m_0 m_1 \dots m_k$. We define the probability of the cylinder set by $\text{Prob}^{\mathcal{M}}(\text{Cyl}(m_0 m_1 \dots m_k)) := \mu_0(m_0) \prod_{i=0}^{k-1} P(m_i, m_{i+1})$. For a given MC $\mathcal{M}$, $\text{Prob}^{\mathcal{M}}$ can be uniquely extended to a probability measure over the σ -algebra generated by all cylinder sets.

In this paper we are interested in ω -regular properties $L \subseteq \Sigma^\omega$ and the probability $\text{Prob}^{\mathcal{M}}(L)$ for some measurable set L . Further, we define $\text{Prob}^{\mathcal{M}}(\mathcal{B}) := \text{Prob}^{\mathcal{M}}(\{\xi \in \text{Paths}^{\mathcal{M}} \mid \xi \in \mathcal{B}\})$ for an automaton $\mathcal{B}$. We write $\text{Prob}_m^{\mathcal{M}}$ to denote the probability function when assuming that m is the initial state. Moreover, we omit the superscript $\mathcal{M}$ whenever it is clear from the context.

We follow the standard way of computing this probability in the product of $\mathcal{M}$ and a deterministic automaton for L . For this purpose we first define product Markov chains that result from the product of the Markov chain and DRA or TGBA, which therefore have accepting conditions on their edges.

Definition 1 (RMC and GMC). Given an MC $\mathcal{M} = (M, L, \mu, P)$ and a deterministic automaton $\mathcal{D} = (\Sigma, Q, q_0, \delta, \text{ACC})$, the product automaton is defined by $\mathcal{M} \times \mathcal{D} := (M \times Q, L', \mu', P', \text{ACC}')$ where

- $P'((m, d), (m', d'))$ equals $P(m, m')$ if $d' = \delta(d, L(m'))$, and is 0 otherwise;
- $L'((m, d)) = L(m)$;
- $\mu'((m, d)) = \mu(m)$ if $d = \delta(q_0, L(m))$, and 0 otherwise;
- if $\text{ACC} = \{F_i \mid i \in [1..k]\}$, then $\text{ACC}' := \{F'_i \mid i \in [1..k]\}$ where $F'_i = \{((m, d), (m', d')) \mid (d, L(m'), d') \in F_i\}$ (Generalised Büchi Markov chain, GMC);

– if $\mathcal{ACC} = \{(A_i, R_i) \mid i \in [1..k]\}$, then $\mathcal{ACC}' := \{(A'_i, R'_i) \mid i \in [1..k]\}$ where $A'_i = \{((m, d), (m', d')) \mid (d, L(m'), d') \in A_i\}$ and $R'_i = \{((m, d), (m', d')) \mid (d, L(m'), d') \in R_i\}$ (Rabin Markov chain, RMC).

Thus, the product automaton is an MC with corresponding accepting conditions. Note that we omitted labels of the transitions in the accepting transitions as they can be read from the label of the states emitting the transitions. We remark that the labelling of the initial states of the Markov chain is taken into account in the definition of μ' .

A leaf strongly connected component (SCC) $S \subseteq V$ is an SCC in the underlying digraph (V, E) of an MC, where all edges with source in S have successors in S (i.e., $\forall (s, s') \in E. s \in S$ implies $s' \in S$). We assume that a (leaf) SCC does not contain any absorbing states.

3 Lazy Determinisation

We fix an input MC $\mathcal{M}$ and a TGBA $\mathcal{B} = (\Sigma, Q, I, T, \{F_i \mid i \in [1..k]\})$ as a specification. Further, let $\mathcal{A} = \det(\mathcal{B})$ be the deterministic Rabin automaton (DRA) constructed for $\mathcal{B}$, and let $\mathcal{M} \times \mathcal{A} := (M \times Q, L, \mu, P, \mathcal{ACC})$ be the product RMC.

We consider the problem of computing $\text{Prob}_{m_0}^{\mathcal{M}}(\mathcal{B})$, i.e., the probability that a run of $\mathcal{M}$ is accepted by $\mathcal{B}$.

3.1 Outline of our methodology

We first recall the classical approach for computing $\text{Prob}^{\mathcal{M}}(\mathcal{B})$, see [BK08]. It is well known [BdA95] that the computation of $\text{Prob}^{\mathcal{M}}(\mathcal{B})$ reduces to the computation of the probabilistic reachability in the product RMC. We need first the notion of accepting SCCs:

Lemma 1. *Let S be a leaf SCC of the product RMC. We say S is accepting iff there is an index i such that S contains some edge in A_i , but no edge in R_i . Formally, there exists $(A_i, R_i) \in \mathcal{ACC}$ such that $A_i \cap \{(d, L(m'), d') \mid P|_S((m, d), (m', d')) > 0\} \neq \emptyset$ and $R_i \cap \{(d, L(m'), d') \mid P|_S((m, d), (m', d')) > 0\} = \emptyset$. Here $P|_S$ denotes the probabilistic transition restricted to the set S .*

Moreover, we say s is accepting if there exists an accepting leaf SCC S with $s \in S$.

Essentially, if a path ends in a state $s \in S$, then almost all of its continuations are accepting (or rejecting) if S is accepting (or rejecting), that is they are accepting (rejecting) with probability 1. Now we have the following reduction:

Theorem 1 ([BdA95]). *Let $U \subseteq M \times Q$ denote the set of accepting states and let $\Diamond U$ denote the set of paths containing a state of U . Then, $\text{Prob}^{\mathcal{M}}(\mathcal{B}) = \text{Prob}^{\mathcal{M} \times \mathcal{A}}(\Diamond U)$.*

When all leaf SCCs are evaluated, the evaluation of the Rabin MC is simple: we abstract all accepting leaf SCCs to an absorbing goal state and perform a reachability analysis, which can be solved in polynomial time. Thus, the outline of the traditional probabilistic model checking approach for LTL specifications is as follows:

1. translate the TGBA $\mathcal{B}$ into an equivalent DRA $\mathcal{A} = \det(\mathcal{B})$;
2. build (the reachable fragment of) the product automaton $\mathcal{M} \times \mathcal{A}$;
3. for each leaf SCC S , check whether S is accepting. Let U be the union of these accepting SCCs;

4. infer the probability $\text{Prob}^{\mathcal{M} \times \mathcal{A}}(\Diamond U)$.

The deterministic Rabin automaton used in the classical approach is often the bottleneck of the approach, as one exploits some variant of the approach proposed by Safra [Saf88], which is rather involved. The lazy determinisation approach we suggest in this paper follows a slightly different approach. Instead of using $\det(\mathcal{B})$ directly, we construct two deterministic TGBAs $\mathcal{B}^u, \mathcal{B}^o$ directly from the subset construction, such that $\mathcal{L}(\mathcal{B}^u) \subseteq \mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{B}^o)$ holds. Thus, $\mathcal{B}^u, \mathcal{B}^o$ under- and over-approximate $\mathcal{B}$, respectively. We use them to build product GMCs and get lower- and upper-bound for the probability $\text{Prob}^{\mathcal{M} \times \mathcal{A}}(\Diamond U)$.

The approach is *incremental* in the sense that if the bound is not tight, we refine the product GMC instead of restarting. In more detail, we identify only the leaf SCCs which cause the imprecise bound, and use a refined breakpoint-automaton there. If both fail to examine that leaf SCC, we resort to the precise Rabin automaton.

In next subsection we first provide the theoretical background, then we present the incremental evaluation of the leaf SCCs.

3.2 Acceptance Equivalence

In order to be able to apply the determinisation lazily, we will exploit a number of acceptance equivalences in the RMC.

For a given DRA $\mathcal{A}$, we denote by $\mathcal{A}_d$ the otherwise similar DRA with initial state d . Similarly, for a TGBA $\mathcal{B}$, we denote by $\mathcal{B}_A$ the TGBA with the set of initial states A .

Theorem 2. *Let d be an arbitrary state of $\mathcal{A}$. Then, a word is accepted by $\mathcal{A}_d$ iff it is accepted by $\mathcal{B}_{\text{reach}(d)}$.*

The proof is a minor variation of the acceptance proof of the DRA construction [Saf88]. Theorem 2 provides two immediate corollaries.

Corollary 1. *For an MC $\mathcal{M}$ and DRA $\mathcal{A} = \det(\mathcal{B})$, a path ρ in $\mathcal{M} \times \mathcal{A}$ that starts from a state (m, d) is accepted iff the word it defines is accepted by $\mathcal{B}_{\text{reach}(d)}$.*

Corollary 2. *For an MC $\mathcal{M}$ and DRA $\mathcal{A} = \det(\mathcal{B})$, if $\text{reach}(d) = \text{reach}(d')$, then the probabilities of acceptance from a state (m, d) and a state (m, d') are equal, i.e., $\text{Prob}_{(m,d)}^{\mathcal{M} \times \mathcal{A}}(\mathcal{B}) = \text{Prob}_{(m,d')}^{\mathcal{M} \times \mathcal{A}}(\mathcal{B})$.*

This property allows us to work on quotients *and* to swap (finitely many times) between states with the same reachability set. The quotient $[d]$ of a state d of a DRA $\mathcal{A} = \det(\mathcal{B})$ is $[d] = \text{reach}(d)$. By abuse of notation, we use $[(m, d)] = (m, [d])$ and $[S] = \{[s] \mid s \in S\}$. It is straightforward to check that the quotient is well defined: for $(m, d_1), (m, d_2) \in [(m, d)]$, it holds $P((m, d_1), [(m', d')]) = P((m, d_2), [(m', d')]) = P(m, m')$. Thus the equivalence induces an MC, which will be referred to as the *quotient MC* below.

Using the quotient MC, the relation between paths and leaf SCCs of the product DRA becomes apparent.

Lemma 2. *For an MC $\mathcal{M}$, a DRA $\mathcal{A} = \det(\mathcal{B})$, and the RMC $\mathcal{R} = \mathcal{M} \times \mathcal{A}$, it holds:*

1. *if there is a path from (m, d) to (m', d') in $\mathcal{M} \times \mathcal{A}$, then there is a (similar) path from $[(m, d)]$ to $[(m, d')]$ in the quotient MC, and*

2. if there is a path from $[(m, d)]$ to $[(m', d')]$ and (m, d) is reachable in $\mathcal{M} \times \mathcal{A}$, then there is a (similar) path from (m, d) to some (m, d'') with $\text{reach}(d') = \text{reach}(d'')$ in $\mathcal{M} \times \mathcal{A}$.

In particular, if $[(m, d)]$ is reachable in the quotient MC, then (m, d') is reachable in $\mathcal{M} \times \mathcal{A}$ for some d' with $\text{reach}(d) = \text{reach}(d')$.

Both of the claims are easy to establish by induction over the length of the path. They prepare the relevant theorem about leaf SCCs.

Lemma 3. For an MC $\mathcal{M}$ and DRA $\mathcal{A} = \det(\mathcal{B})$, it holds that

1. if S is a leaf SCC in $\mathcal{M} \times \mathcal{A}$ then $[S]$ is a leaf SCC of the quotient MC,
2. if S' is a leaf SCC in the quotient MC, then there is a leaf SCC S in $\mathcal{M} \times \mathcal{A}$ with $S' = [S]$.

Together with Lemma 1 and Theorem 2, Lemma 3 provides:

Corollary 3. Let S be a leaf SCC of the quotient MC. Then, either all states s of $\mathcal{M} \times \mathcal{A}$ with $[s] \in S$ are accepting, or all states s of $\mathcal{M} \times \mathcal{A}$ with $[s] \in S$ are rejecting.

Thus, the value of nodes in leaf SCCs of the quotient MC is qualitative, and any technique that allows us to establish the value of some node in a leaf SCC will suffice to determine the value of all nodes in the leaf SCC. Once all leaf SCCs are evaluated, we need only to perform a standard probabilistic reachability analysis on the quotient MC, which can be solved in polynomial time, see for example [BK08] for details.

3.3 Incremental Evaluation of Leaf SCCs

To evaluate each leaf SCC of the RMC, we suggest to use three techniques: the first one is based on evaluating the subset construction directly. We get two deterministic TGBAs that provide over- and under-approximations. If this fails, we refine the corresponding leaf SCC by a breakpoint construction. Only if both fail, a precise determinisation step follows.

Estimation of accepting transitions. For a given TGBA $\mathcal{B} = (\Sigma, Q, I, T, \{F_i \mid i \in [1..k]\})$, a simple way to over- and underapproximate its language by a subset construction is as follows. We build two TGBAs, $\mathcal{S}^o = (\Sigma, 2^Q, \{I\}, T', \{F_i^o \mid i \in [1..k]\})$ and $\mathcal{S}^u = (\Sigma, 2^Q, \{I\}, T', \{F_i^u \mid i \in [1..k]\})$ with

$$\begin{aligned} - T' &= \{(R, \sigma, C) \mid C = \{q' \in Q \mid \exists q \in R. (q, \sigma, q') \in T\} \neq \emptyset\}, \\ - F_i^o &= \{(R, \sigma, C) \in T' \mid \exists (q, q') \in R \times C. (q, \sigma, q') \in F_i\}, \text{ and} \\ - F_i^u &= \{(R, \sigma, C) \in T' \mid \nexists (q, q') \in R \times C. (q, \sigma, q') \in T \setminus F_i\}. \end{aligned}$$

Essentially, $\mathcal{S}^o$ and $\mathcal{S}^u$ are subset automata where we over- and under-approximate the accepting conditions, respectively. The following lemma holds:

Lemma 4. $\mathcal{L}(\mathcal{S}_{[d]}^u) \subseteq \mathcal{L}(\mathcal{A}_d) \subseteq \mathcal{L}(\mathcal{S}_{[d]}^o)$.

The proof is easy as in F_i^o and F_i^u the accepting transitions are over- and under-approximated. The proof will be established as a part of Theorem 3 later. With this lemma, we are able to identify some accepting and rejecting leaf SCCs in the product.

We remark that $\mathcal{S}^o, \mathcal{S}^u$ are the same except for their accepting conditions. Thus, the corresponding GMCs $\mathcal{M} \times \mathcal{S}^u$ and $\mathcal{M} \times \mathcal{S}^o$ are also the same except for their accepting conditions. Moreover: this GMC is precisely the quotient Markov chain we discussed in the previous subsection. With the accepting conditions, we can identify some accepting or rejecting leaf SCCs as below:

Lemma 5. *Let $\mathcal{M}$ be an MC and $\mathcal{B}$ a TGBA. Let $\mathcal{S}^o, \mathcal{S}^u$ as defined above. Let S be a leaf SCC of $\mathcal{M} \times \mathcal{S}^u$. Then,*

- *S is accepting if $F_i^u \cap \{(R, L(m'), C) \mid P|_S((m, R), (m', C)) > 0\} \neq \emptyset$ holds for all $i \in [1..k]$;*
- *S is rejecting if $F_i^o \cap \{(R, L(m'), C) \mid P|_S((m, R), (m', C)) > 0\} = \emptyset$ holds for some $i \in [1..k]$.*

Thus the above result directly follows by Lemma 4. In case there are remaining leaf SCCs for which we cannot conclude whether they are accepting or rejecting, we continue with the breakpoint construction.

Breakpoint construction. For a given TGBA $\mathcal{B} = (\Sigma, Q, I, T, \{F_i \mid i \in [1..k]\})$, a further simple way of over- and underapproximate its language is by a breakpoint construction. We denote with $\text{breakpt}(Q, k) = \{(R, i, C) \mid C \subsetneq R \subseteq Q \text{ and } i \in [1..k]\}$ the breakpoint set, where R and i intuitively refer to the set of reachable states in the root of an extended history tree and the index of the root, respectively, and C is the set the children of the root are labelled with. That is, we build two deterministic Rabin automata, called the breakpoint automata, $\mathcal{B}^o = (\Sigma, \text{breakpt}(Q, k), (I, 1, \emptyset), T'', \{(A_\varepsilon, \emptyset), (T'', R_1)\})$ and $\mathcal{B}^u = (\Sigma, \text{breakpt}(Q, k), (I, 1, \emptyset), T'', \{(A_\varepsilon, \emptyset)\})$, with

- $A_\varepsilon = ((R, i, C), \sigma, (R', i \oplus_k 1, \emptyset)) \mid i \in [1..k], (R, \sigma, R') \in T', R' \neq \emptyset \text{ and } \exists C' \subseteq Q. (C, \sigma, C') \in T' \text{ such that } (R' \cap F_i) \cup C' = R'$,
- $R_1 = \{((R, i, C), \sigma, (R', i, R' \cap F_i)) \mid i \in [1..k], (R, \sigma, R') \in T', (C, \sigma, \emptyset) \in T', \text{ and either } R' \not\subseteq F_i \text{ or } R' = \emptyset\}$,
- $N = \{((R, i, C), \sigma, (R', i, C')) \mid i \in [1..k], (R, \sigma, R') \in T', R' \neq C', \text{ and } \exists C'' \subseteq Q. (C, \sigma, C'') \in T' \text{ such that } C' = (R' \cap F_i) \cup C'' \text{ and } C'' \neq \emptyset\}$, and
- $T'' = A_\varepsilon \cup R_1 \cup N$.

In a triple (R, i, C) (R for root and C for children, respectively), R keeps the set of reachable states represented in the root of the tree, i the set of final states currently flagged in the root, and C keeps the union of the states represented by its children. We refer to R by $\text{set}((R, i, C))$, to i by $\text{index}((R, i, C))$, and to C by $\text{children}((R, i, C))$.

These triples can also be viewed as quotients of extended history trees. For an extended history tree $d = (\mathcal{T}, l, h)$, we write $\langle d \rangle = (l(\varepsilon), h(\varepsilon), \bigcup_{i \in \mathcal{T} \cap \mathbb{N}} l(i))$, where $\mathcal{T} \cap \mathbb{N}$ denotes the indices of children of ε .

Casting aside the acceptance mechanism, we observe, for $\mathcal{A} = \det(\mathcal{B})$, that the run ρ of $\mathcal{B}_{\langle d \rangle}^o$ and $\mathcal{B}_{\langle d \rangle}^u$, the run ρ' of $\mathcal{A}_d$, and the run ρ'' of $\mathcal{S}_{[d]}^o$ and $\mathcal{S}_{[d]}^u$ on an input word α

- $\text{set}((\rho(n))) = \text{reach}((\rho'(n))) = \rho''(n)$,
- $\text{children}((\rho(n))) = \bigcup_{i \in \mathcal{T}_n \cap \mathbb{N}} l_n(i)$, and
- $\text{index}((\rho(n))) = h_n(\varepsilon)$

hold for $\rho'(n) = (\mathcal{T}_n, l_n, h_n)$. This allows us to establish the following theorem (the proofs are moved to the appendix).

Theorem 3. $\mathcal{L}(\mathcal{S}_{[d]}^u) \subseteq \mathcal{L}(\mathcal{B}_{\langle d \rangle}^u) \subseteq \mathcal{L}(\mathcal{A}_d) \subseteq \mathcal{L}(\mathcal{B}_{\langle d \rangle}^o), \mathcal{L}(\mathcal{S}_{[d]}^o)$.

First we remark that the breakpoint construction can be refined further such that it is finer than $\mathcal{L}(\mathcal{S}_{[d]}^o)$. However we leave it as further work to avoid heavy technical preparations. Exploiting the above theorem, the following becomes clear.

Corollary 4. *Let S be a leaf SCC of the quotient MC. Let $(m, R) \in S$ be an arbitrary state of S and $d = (m, (R, 1, \emptyset))$. Moreover, let $\mathcal{B}_{\langle d \rangle}^o, \mathcal{B}_{\langle d \rangle}^u$ be the breakpoint automata. Then,*

- S is accepting if there exists a leaf SCC S' in $\mathcal{M} \times \mathcal{B}_{\langle d \rangle}^u$ with $S = [S']$ which is accepting (i.e. S' contains some transition in A_ε).
- S is rejecting if there exists a leaf SCC S' in $\mathcal{M} \times \mathcal{B}_{\langle d \rangle}^o$ with $S = [S']$ which is rejecting (i.e. S' contains no transition in A_ε , but some transition in R_1).

Note that the products $\mathcal{M} \times \mathcal{B}_{\langle d \rangle}^u$ and $\mathcal{M} \times \mathcal{B}_{\langle d \rangle}^o$ are the same RMCs except for their accepting conditions. Thus the above result directly follows from Theorem 3. In case there are remaining leaf SCCs for which we cannot conclude whether they are accepting or rejecting, we continue with the Rabin construction.

Together with Corollary 3, Lemma 5 and Corollary 4 immediately provide the following corollary, which justifies the incremental evaluations of the leaf SCCs.

Corollary 5. *If $\mathcal{A} = \det(\mathcal{B})$, $[(m, d)]$ is a state in a leaf SCC of the quotient MC, and $[d] = [d']$, then*

- $\text{Prob}_{(m, d)}^{\mathcal{M} \times \mathcal{A}_d}(\mathcal{B}) = 1$ if $\text{Prob}_{(m, [d])}^{\mathcal{M} \times \mathcal{S}_{[d]}^u}(\mathcal{B}) > 0$ or $\text{Prob}_{(m, \langle d \rangle)}^{\mathcal{M} \times \mathcal{B}_{\langle d \rangle}^u}(\mathcal{B}) > 0$, and
- $\text{Prob}_{(m, d)}^{\mathcal{M} \times \mathcal{A}_d}(\mathcal{B}) = 0$ if $\text{Prob}_{(m, [d])}^{\mathcal{M} \times \mathcal{S}_{[d]}^o}(\mathcal{B}) < 1$ or $\text{Prob}_{(m, \langle d \rangle)}^{\mathcal{M} \times \mathcal{B}_{\langle d \rangle}^o}(\mathcal{B}) < 1$.

4 Markov Decision Processes

The lazy determinisation approach proposed in this paper extends to Markov decision processes (MDP) after minor adaptation; Markov chains have mainly been used for ease of notation. While the details of the extension to MDPs have been moved to Appendix D, we here give an outline of the adaptation with a focus on the differences and particularities that need to be taken into consideration when we are dealing with MDPs.

An MDP is a tuple $\mathcal{M} = (M, \Sigma, \mu_0, \text{Steps})$ with $\text{Steps}: M \times \Sigma \rightarrow \text{Dist}(M)$. The nondeterministic choices are resolved by a scheduler that chooses the next action to be executed depending on a finite path. Like for Markov chains, the principle technique to analyse MDPs against a specification φ is to construct a deterministic Rabin automaton $\mathcal{A}$, build the product $\mathcal{M} \times \mathcal{A}$, and analyse it. This product will be referred to as a *Rabin MDP* (RMDP). For an RMDP, it suffices to consider memoryless deterministic schedulers of the form $\delta: M \times D \rightarrow \Sigma$, where D are the states of $\mathcal{A}$. Given a TGBA specification $\mathcal{B}_\varphi$, we are interested in $\sup_\delta \text{Prob}^{\mathcal{M}, \delta}(\mathcal{B}_\varphi)$. In particular, one can use finite state schedulers on $\mathcal{M}$. (Schedulers that control $\mathcal{M}$ can be used to control $\mathcal{M} \times \mathcal{A}$ for all deterministic automata $\mathcal{A}$.) The superscript $\mathcal{M}$ is omitted when it

is clear from the context. We remark that the infimum can be treated accordingly, as $\inf_{\delta} \text{Prob}^{\delta}(\mathcal{B}_{\varphi}) = 1 - \sup_{\delta} \text{Prob}^{\delta}(\mathcal{B}_{\neg\varphi})$.

As Theorem 2 operates on the word level, Corollary 2 immediately extends to MDPs. Under the corresponding equivalence relation we obtain a *quotient MDP*. Consequently, we can extend the probability to achieve the objective from a state (under optimal control) to quotients, and hence to refer to the probability to achieve the control objective from a state in a quotient MDP of the RMDP.

From here, it is apparent that we can use the estimation of the word languages provided in Theorem 3 to estimate $\sup_{\delta} \text{Prob}^{\delta}(\mathcal{B}_{\varphi})$.

Corollary 6. *Let m be a state of $\mathcal{M}$, let d be a state of $\mathcal{A}$. Then $\sup_{\delta} \text{Prob}_{(m,[d])}^{\delta}(\mathcal{S}_{[d]}^u) \leq \sup_{\delta} \text{Prob}_{(m,\langle d \rangle)}^{\delta}(\mathcal{B}_{\langle d \rangle}^u) \leq \sup_{\delta} \text{Prob}_{(m,d)}^{\delta}(\mathcal{A}_d) \leq \sup_{\delta} \text{Prob}_{(m,\langle d \rangle)}^{\delta}(\mathcal{B}_{\langle d \rangle}^o), \sup_{\delta} \text{Prob}_{(m,[d])}^{\delta}(\mathcal{S}_{[d]}^o)$ holds.*

The inequalities are apparent, as the same control strategy for the underlying MDP can be used. (Note that a control strategy for $M \times \mathcal{A}_d$ is not necessarily memoryless for $\mathcal{S}_d$ or $\mathcal{B}_{\langle d \rangle}$, but they are all finite memory controllers for M itself.)

In the standard evaluation of RMDP, the *end components* of the product $\mathcal{M} \times \mathcal{A}$ play a role comparable to the one played by leaf SCCs in MCs. An end component is simply a sub-MDP, which is closed in the sense that there exists a memoryless scheduler δ such that the induced Markov chain is a leaf SCC. Thus, one can stay in the end component and traverse all of its transitions infinitely often, where acceptance is defined as for leaf SCCs in Markov chains. For RMDPs, it suffices to analyse maximal end components, see [BK08] for details.

The basic technique is to determine these maximal end components. They are easy to construct and, for each accepting pair, they are easy to evaluate: it suffices to remove the rejecting transitions, repeat the construction of MECs on the remainder, and check if there is any that contains an accepting transition.

Once the accepting MECs are determined, their states are assigned a winning probability of 1, and evaluating the complete MDP reduces to a maximal reachability analysis, which in turn reduces to solving a linear programming problem and can thus be solved in polynomial time.

Now we argue that a layered approach of lazy determinisation can be followed for MDPs, which is rather comparable to the one described for Markov chains. We start with the quotient MDP, and consider an arbitrary MEC S . Using accepting conditions of the subset automata $\mathcal{S}^u$ and $\mathcal{S}^o$, we check whether S is accepting and rejecting, respectively. If this test is inconclusive, then we first refine S by a breakpoint construction, and finally a Rabin automaton construction (full determinisation). We remark that, as for Markov chains, the breakpoint and Rabin constructions can be considered as oracles: when we have identified the accepting MECs, a plain reachability analysis is performed on the quotient MDP. For details on the evaluation of MDPs, we refer to Appendix D.

5 Implementation and Results

We have implemented our approach in a prototypical tool. The automaton as well as the model and the product are stored explicit-state wise. We compare our tool with the model checker PRISM and another recent prototype tool used in [CGK13] developed for a subclass of LTL formulas. To do the comparison, we use LTL formulas for specifying

properties. Our experimental results suggest that our tool provides a practical approach for checking LTL properties for probabilistic systems. The tool and more examples can be found under: <http://lcs.ios.ac.cn/~zhanglj/lazy/>

We first argue that our approach can be directly used to check LTL specifications.

5.1 Lineal Temporal Logic

For an LTL formula φ , Vardi and Wolper [VW86] have shown that one can construct a Büchi automaton $\mathcal{B}_\varphi$ such that $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{B}_\varphi)$. One can apply off-the-shelf established tools like LTL3BA [BKRS12] or SPOT [DL11] to translate an LTL specification to a TGBA. We remark that it is well-known that TGBAs are more expressive than LTL formulas.

For Markov chains, one can translate both φ and $\neg\varphi$ into equivalent TGBAs $\mathcal{B}_\varphi$ and $\mathcal{B}_{\neg\varphi}$, respectively. If one of them is deterministic, it can be used directly; otherwise we just select $\mathcal{B}$ to be the smaller one of both to continue. The reason for creating both, the automaton for φ and for $\neg\varphi$, is that the complementation of Büchi [Sch09a, Yan08] and generalised Büchi automata [SV12] can lead to a factorial increase in their size. It is therefore a realistic assumption that the difference in the size of $\mathcal{B}_\varphi$ and $\mathcal{B}_{\neg\varphi}$ can be significant. As the translation to nondeterministic TGBA is believed to be comparably cheap, we deem this step worth taking. Indeed, the runtime for this transformation using SPOT was negligible in all experiments we have conducted.

5.2 Experimental Results

We applied our implementation on several models exported from the probabilistic model checker PRISM. For all experiments, we used a 2.9 GHz Intel Core i7 with 8 GB 1600 MHz DDR3 RAM. We aborted tool runs when they took more than 30 minutes or needed more than 4GB of RAM.

Workstation cluster [HHK00]. We consider two clusters of $n = 16$ workstations each, so that the two clusters are connected by a backbone. Each of the workstations may fail with a given rate, as may the backbone. Though this case study is a continuous-time Markov chain, we focused on time-unbounded properties, such that we could use discrete-time Markov chains to analyse them. The properties $prop\mathbf{U}_k$ are probabilities of the event of component failures with respect to the order (first k failures on left before right) while $prop\mathbf{GF}\wedge_k$ and $prop\mathbf{GF}\vee_k$ describe the long-run number of workstations functional.

Results are given in Table 1. We give results for our prototypical tool (“prototype”) as well as for PRISM (“PRISM”). We depict the probability computed (“prob.”), the total runtime in seconds (“time”), the number of states reached in the product of the model and the automaton (“states”) and the number of states of the automaton, which were constructed (“aut.”). For our prototype, we state the determinisation procedure, with which we were able to obtain a conclusive result (“type”). The number of automata states refers to this automata type. For PRISM, we used the sparse matrix engine, because it has proven to be the fastest. For the model checking of general LTL properties, PRISM first transforms the formulas into Büchi automata, which are subsequently transformed into deterministic Rabin automata.

As we see, for this case study our approach is faster than the one of PRISM for larger model instantiations. The reasons for this is as follows: for this case study, for the

property	prototype				PRISM			
	prob.	time	states	aut. type	prob.	time	states	aut.
$prop\mathbf{U}_7$	0.509641789089	1	10836	9 subset	0.509641789089	0	20480	10
$prop\mathbf{U}_8$	0.509641789162	2	10928	10 subset	0.509641789162	1	20508	11
$prop\mathbf{U}_9$	0.509641789168	2	11020	11 subset	0.509641789168	6	20536	12
$prop\mathbf{U}_{10}$	0.509641789168	2	11020	11 subset	0.509641789168	24	20536	13
$prop\mathbf{U}_{11}$	0.509641789168	3	11020	11 subset	-	-	-	-
$prop\mathbf{GF}\wedge_2$	0	5	13704	4 breakp.	0	1	12036	1609
$prop\mathbf{GF}\wedge_3$	0	3	15028	5 breakp.	0	61	12724	30537
$prop\mathbf{GF}\wedge_4$	0	3	16352	6 breakp.	-	-	-	-
$prop\mathbf{GF}\vee_2$	1	3	27625	11 rabin	1	2	12748	824
$prop\mathbf{GF}\vee_3$	1	3	30413	14 rabin	1	278	13360	14168
$prop\mathbf{GF}\vee_4$	1	4	33201	17 rabin	-	-	-	-

Table 1. Results for the workstation cluster. Above, we let

$prop\mathbf{U}_k := \mathcal{P}_{=?}(\text{left_}n=n \mathbf{U} (\text{left_}n=n-1 \mathbf{U} (\dots \mathbf{U} (\text{left_}n=n-k \mathbf{U} \text{right_}n \neq n) \dots)),$
 $prop\mathbf{GF}\wedge_k := \mathcal{P}_{=?}(\mathbf{GF}\text{left_}n=n \wedge \bigvee_{i=0}^k \mathbf{FG}\text{right_}n=n-i),$ and
 $prop\mathbf{GF}\vee_k := \mathcal{P}_{=?}(\mathbf{GF}\text{left_}n=n \vee \bigvee_{i=0}^k \mathbf{FG}\text{right_}n=n-i).$

properties $prop\mathbf{U}_k$ the subset is sufficient, as both under and overapproximation lead to the same result. For $prop\mathbf{GF}\wedge_k$, the breakpoint construction suffices. Thus, we can use computationally cheaper algorithms than the one implemented in PRISM. Indeed, PRISM spends by far most of the runtime for the computation of the deterministic Rabin automata. For $prop\mathbf{GF}\vee_k$, we had to use the full Rabin determinisation. However, our prototype was still faster than PRISM here, because we compute states of the Rabin automaton on the fly and thus never had to construct the complete automaton.

Self-stabilising algorithm [IJ90]. In this case study, we consider a number of n processes connected by a ring-formed network. Initially, one or more processes have a token. The case study considers a self-stabilising algorithm that targets at reducing the number of tokens to exactly one, which can then be passed around the network to uniquely identify some privileged process. To do so, because of the ring-formed structure of the network, processes can only communicate with their direct neighbours. The variant we consider is an MDP. For different n , we consider a number of properties about the order, in which the number of tokens decreases.

We give results in Table 2. The columns given are as in the last case studies. In this case study, the subset construction is sufficient for all model checking runs. Performance figures of our tool are better than those for PRISM, because we spend significantly less effort in constructing automata states.

Randomised mutual exclusion [PZ86]. We consider a mutual exclusion protocol together with a set of properties analysed previously in [CGK13]. The approach proposed in [CGK13] involves a specialised algorithm for LTL formula without the Until and Next but including the Finally and Globally operators. Instead of comparing to PRISM, we compare to the numbers given in this previous work. The comparison cannot be completely accurate, because their implementation is not publicly available. Also, for their results they just state that they used an Intel i7 with 8 GB RAM but did not discuss the exact speed of the machine. From comparison of PRISM runtimes (not given in the ta-

property	n	prototype			PRISM			
		prob.	time	states aut.	prob.	time	states aut.	
$prop_4$	4	1	0	39	4	1	0	25 161
	5	1	1	85	4	1	0	52 161
	6	1	0	175	4	1	0	106 161
	7	1	1	352	4	1	0	214 161
$prop_5$	5	1	1	86	5	1	1	56 1822
	6	1	1	189	5	1	1	114 1822
	7	1	1	387	5	1	1	230 1822
$prop_6$	6	1	1	181	6	1	51	119 24824
	7	1	1	401	6	1	52	240 24824
$prop_7$	7	1	2	372	7	-	-	- -

Table 2. Results for self-stabilising algorithm.

Above, we let $prop_k := \mathcal{P}_{\min=?}(\mathbf{G}(t=k \rightarrow (\dots (t=k \mathbf{U} t=k-1) \mathbf{U} t=1) \mathbf{U} \dots t=1) \dots)$.

ble) we assume that the machine we use is about 11 to 18 percent faster. This case study is an MDP.

In Table 3 we give a comparison of our implementation with the specialised algorithm from [CGK13]. For this case study, we did not use the breakpoint construction, as it did not lead to any improvement in runtime or memory usage. As we see, in most very large cases, the approach of [CGK13] is faster than ours. This was to be expected, as the previous approach is an algorithm particularly tuned for the type of properties considered: the total complexity of the translation of LTL formulas of size n to deterministic automata used in [CGK13] is in $2^{O(2^n)}$ whereas the total complexity of the translation from LTL to Rabin automata is in $2^{O(2^{n+\log_2 n})}$ [KE12]. However, for medium-sized model instances and also some larger analyses the runtimes of our tool are comparable or better. For property (9), we remark that our approach is much slower when using the incremental approach to decide components. However, for this particular property, when we just use the breakpoint construction directly, the results are conclusive. Then, for $n = 3$ we obtain a runtime of 9 seconds while for $n = 4$ we get a runtime of 128 seconds.

Summary. For the cases considered above, except for very small models or formulas, our prototype outperforms PRISM. In particular, this holds for the cases where we were able to use the subset or breakpoint determinisation successfully. However, because we use an on-the-fly approach that only expands inconclusive leafs, our method was faster even if we had to fall back to the complete Rabin determinisation in some parts of the model. When comparing with the specialised approach of Chatterjee et al. [CGK13], we achieve comparable or better times for models of medium size, while their tool is mostly faster for very large models. However, our prototype was faster in some larger cases even than their specialised algorithm.

6 Conclusion

We have proposed a lazy determinisation approach for analysing Markov models against specifications given by Büchi automata or LTL formulas. We have implemented the ap-

property	n	prob. time	prototype			[CGK13] time
			states	aut.	type	
$\mathcal{P}_{\max=?}(\mathbf{GF} p_1=10$	3	1	2	2368	1 subset	0
$\wedge \mathbf{GF} p_2=10$	(1) 4	1	3	27600	1 subset	0
$\wedge \mathbf{GF} p_3=10)$	5	1	13	308800	1 subset	0
$\mathcal{P}_{\max=?}(\mathbf{GF} p_1=10 \wedge \mathbf{GF} p_2=10$	4	1	4	27600	1 subset	0
$\wedge \mathbf{GF} p_3=10 \wedge \mathbf{GF} p_4=10)$	(2) 5	1	19	308800	1 subset	0
$\mathcal{P}_{\min=?}(\mathbf{GF} p_1=10 \wedge \mathbf{GF} p_2=10$	4	0	25	505239	62 rabin	36
$\wedge \mathbf{GF} p_3=10 \wedge \mathbf{GF} p_4=10)$	(3) 5	-	-	-	-	607
$\mathcal{P}_{\max=?}((\mathbf{GF} p_1=0 \vee \mathbf{FG} p_2 \neq 0)$	3	1	5	19981	33 rabin	2
$\wedge (\mathbf{GF} p_2=0 \vee \mathbf{FG} p_3 \neq 0))$	(4) 4	1	16	222635	32 rabin	29
	5	-	-	-	-	479
$\mathcal{P}_{\max=?}((\mathbf{GF} p_1=0 \vee \mathbf{FG} p_2 \neq 0)$	3	1	5	15594	20 rabin	2
$\wedge (\mathbf{GF} p_2=0 \vee \mathbf{FG} p_2 \neq 0))$	(5) 4	1	19	358820	36 rabin	28
	5	-	-	-	-	463
$\mathcal{P}_{\max=?}((\mathbf{GF} p_1=0 \vee \mathbf{FG} p_2 \neq 0)$	3	1	9	57590	96 rabin	7
$\wedge (\mathbf{GF} p_2=0 \vee \mathbf{FG} p_3 \neq 0)$	(6) 4	1	81	674417	96 rabin	75
$\wedge (\mathbf{GF} p_3=0 \vee \mathbf{FG} p_1 \neq 0))$	5	-	-	-	-	1219
$\mathcal{P}_{\max=?}((\mathbf{GF} p_1=0 \vee \mathbf{FG} p_2 \neq 0)$	3	1	17	90297	155 rabin	7
$\wedge (\mathbf{GF} p_2=0 \vee \mathbf{FG} p_2 \neq 0)$	(7) 4	1	136	1042479	155 rabin	78
$\wedge (\mathbf{GF} p_3=0 \vee \mathbf{FG} p_3 \neq 0))$	5	-	-	-	-	1267
$\mathcal{P}_{\min=?}((\mathbf{GF} p_1 \neq 10 \vee \mathbf{GF} p_1=0 \vee \mathbf{FG} p_1=1)$	3	0	5	3021	4 rabin	0
$\wedge \mathbf{GF} p_1 \neq 0 \wedge \mathbf{GF} p_1=1)$	(8) 4	0	5	34995	5 rabin	8
	5	0	30	388771	5 rabin	146
$\mathcal{P}_{\max=?}((\mathbf{G} p_1 \neq 10 \vee \mathbf{G} p_2 \neq 10 \vee \mathbf{G} p_3 \neq 10)$	3	-	-	-	-	80
$\wedge (\mathbf{FG} p_1 \neq 1 \vee \mathbf{GF} p_2=1 \vee \mathbf{GF} p_3=1)$	(9) 4	-	-	-	-	703
$\wedge (\mathbf{FG} p_2 \neq 1 \vee \mathbf{GF} p_1=1 \vee \mathbf{GF} p_3=1))$	5	-	-	-	-	-
$\mathcal{P}_{\min=?}((\mathbf{FG} p_1 \neq 0 \vee \mathbf{FG} p_2 \neq 0 \vee \mathbf{GF} p_3=0)$	3	0	4	12743	40 rabin	3
$\vee (\mathbf{FG} p_1 \neq 10 \wedge \mathbf{GF} p_2=10 \wedge \mathbf{GF} p_3=10)$	(10) 4	0	21	146221	40 rabin	33
	5	-	-	-	-	543

Table 3. Results for randomised mutual exclusion.

proach in a prototypical tool and were able to demonstrate its efficiency on a number of case studies. As future work, we plan to explore how our present explicit-state algorithm can be transformed into a symbolic one, in order to allow for the analysis of very large probabilistic models.

References

- BBC⁺08. Jiri Barnat, Lubos Brim, Ivana Cerná, Milan Ceska, and Jana Tumova. ProbDiVinE-MC: Multi-core ltl model checker for probabilistic systems. In *QEST*, pages 77–78, 2008.
- BdA95. Andrea Bianco and Luca de Alfaro. Model checking of probabalistic and nondeterministic systems. In *FSTTCS*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

- BK08. Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.
- BKRS12. Tomáš Babiak, Mojmír Kretínský, Vojtech Reháč, and Jan Strejcek. LTL to Büchi automata translation: Fast and more deterministic. In *TACAS*, pages 95–109, 2012.
- BLW13. Michael Benedikt, Rastislav Lenhardt, and James Worrell. Ltl model checking of interval markov chains. In *TACAS*, pages 32–46, 2013.
- CB06. Frank Ciesinski and Christel Baier. LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *QEST*, pages 131–132, 2006.
- CGK13. Krishnendu Chatterjee, Andreas Gaiser, and Jan Kretínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In *CAV*, pages 559–575, 2013.
- CY95. Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *JACM*, 42(4):857–907, 1995.
- DL11. Alexandre Duret-Lutz. LTL translation improvements in spot. In *VECoS*, pages 72–83. BCS, 2011.
- HHK00. Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. On the use of model checking techniques for dependability evaluation. In *SRDS*, pages 228–237, 2000.
- HJ94. Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *FAC*, 6(5):512–535, 1994.
- IJ90. Amos Israeli and Marc Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *PODC*, pages 119–131. ACM, 1990.
- KE12. Jan Kretínský and Javier Esparza. Deterministic automata for the (F, G)-fragment of LTL. In *CAV*, pages 7–22, 2012.
- KNP11. Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591, 2011.
- Pit07. Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *JLMCS*, 3(3:5), 2007.
- PZ86. Amir Pnueli and Lenore D. Zuck. Verification of multiprocess probabilistic protocols. *DC*, 1(1):53–72, 1986.
- Saf88. Shmuel Safra. On the complexity of ω -automata. In *FOCS*, pages 319–327. IEEE CSP, 1988.
- Sch09a. Sven Schewe. Büchi complementation made tight. In *STACS*, volume 3 of *LIPI*, pages 661–672. IBFI, 2009.
- Sch09b. Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *FoSSaCS*, volume 5504 of *LNCS*, pages 167–181. Springer, 2009.
- SV12. Sven Schewe and Thomas Varghese. Tight bounds for the determinisation and complementation of generalised Büchi automata. In *ATVA*, volume 7561 of *LNCS*, pages 42–56. Springer, 2012.
- VW86. Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344, 1986.
- Yan08. Qiqi Yan. Lower bounds for complementation of *omega*-automata via the full automata technique. *Journal of Logical Methods in Computer Science*, 4(1:5), 2008.

Appendix

A Proof of Theorem 2

To establish Theorem 2, we show inclusion in both directions. The proof is the same as the correctness proof for the determinisation construction in [SV12], but the claim is different, and the proof is therefore included for completeness. The difference in the claim is that $\mathcal{L}(\mathcal{A}_d) = \mathcal{L}(\mathcal{B}_{\text{reach}(d)})$ is shown for $d = (\mathcal{T}, l, h)$ with a singleton set $\mathcal{T} = \{\varepsilon\}$ that contains only the root and $h(\varepsilon) = 1$. The proof, however, does not use either of these properties.

Lemma 6. $\mathcal{L}(\mathcal{A}_d) \subseteq \mathcal{L}(\mathcal{B}_{\text{reach}(d)})$

Notation. For an ω -word α and $j \geq i$, we denote with $\alpha[i, j[$ the word $\alpha(i)\alpha(i+1)\alpha(i+2)\dots\alpha(j-1)$. We denote with $Q_1 \xrightarrow{\alpha} Q_2$ for a finite word $\alpha = \alpha_1 \dots \alpha_{j-1}$ that there is, for all $q_j \in Q_2$ a sequence $q_1 \dots q_j$ with $q_1 \in Q_1$ and $(q_i, \alpha_i, q_{i+1}) \in T$ for all $1 \leq i < j$. If one of these transitions is guaranteed to be in F_a , we write $Q_1 \xRightarrow{\alpha}_a Q_2$.

For an extended history tree $d = (\mathcal{T}, l, h)$, we denote for a state $q \in \text{reach}(d)$ the node of $\vartheta \in \mathcal{T}$, such that $q \in l(\vartheta)$, but not in any child of ϑ ($\nexists i \in \mathbb{N}. q \in l(\vartheta i)$), by $\text{host}(q, d)$.

For an input word $\alpha : \omega \rightarrow \Sigma$, let $\rho_{\mathcal{D}} = d_0, d_1, \dots$ be the run of a Rabin automaton $\mathcal{D}$ on α . A node v in the history tree d_{i+1} is called *stable* or *accepting*, respectively, if it is stable or accepting in the transition $(d_i, \alpha(i), d_{i+1})$.

Proof. Let $\alpha \in \mathcal{L}(\mathcal{A}_d)$. Then there is a v that is eventually always stable and always eventually accepting in the run $\rho = d_0 d_1 d_2 \dots$ (with $d_0 = d$) of $\mathcal{A}_d$ on α . We pick such a v .

Let $i_0 < i_1 < i_2 < \dots$ be an infinite ascending chain of indices such that

- v is stable for all d_j with $j \geq i_0$, and
- the chain $i_0 < i_1 < i_2 < \dots$ contains exactly those indices $i \geq i_0$ such that d_i is accepting; this implies that h is updated exactly at these indices.

Let $d_i = (\mathcal{T}_i, l_i, h_i)$ for all $i \in \omega$. By construction, we have

- $\text{reach}(d) \xrightarrow{\alpha[0, i_0[} l_{i_0}(v)$, and
- $l_{i_j}(v) \xRightarrow{\alpha[i_j, i_{j+1}[}_{h_{i_j}} l_{i_{j+1}}(v)$.

Exploiting König's lemma, this provides us with the existence of a run that visits all accepting sets F_i of $\mathcal{B}_{\text{reach}(d)}$ infinitely often. (Note that the value of h is circulating in the successive sequences of the run.) This run is accepting, and α therefore in the language of $\mathcal{B}_{\text{reach}(d)}$. $\square$

Lemma 7. $\mathcal{L}(\mathcal{A}_d) \supseteq \mathcal{L}(\mathcal{B}_{\text{reach}(d)})$.

Proof. Let $\alpha \in \mathcal{L}(\mathcal{B}_{\text{reach}(d)})$. We fix an accepting run $\rho = q_0 q_1 \dots$ of $\mathcal{A}$ on an input word α , and let $\rho_{\mathcal{D}} = d_0 d_1 \dots$ be the run of $\mathcal{D}$ on α . We then define the related sequence of host nodes $\vartheta = v_0 v_1 v_2 \dots = \text{host}(q_0, d_0) \text{host}(q_1, d_1) \text{host}(q_2, d_2) \dots$. Let l be the shortest length $|v_i|$ of these nodes that occurs infinitely many times.

We follow the run and see that the initial sequence of length l of the nodes in ϑ eventually stabilises. Let $i_0 < i_1 < i_2 < \dots$ be an infinite ascending chain of indices such that the length $|v_j| \geq l$ of the j -th node is not smaller than l for all $j \geq i_0$, and equal to $l = |v_i|$ for all indices $i \in \{i_0, i_1, i_2, \dots\}$ in this chain. This implies that $v_{i_0}, v_{i_1}, v_{i_2}, \dots$ is a descending chain when the single nodes v_i are compared by lexicographic order. As the domain is finite, almost all elements of the descending chain are equal, say $v_i := \pi$. In particular, π is eventually always stable.

Let us assume for contradicting that this stable prefix π is accepting only finitely many times. We choose an index i from the chain $i_0 < i_1 < i_2 < \dots$ such that π is stable for all $j \geq i$. (Note that π is the host of q_i for d_i , and $q_j \in l_j(\pi)$ holds for all $j \geq i$.)

As ρ is accepting, there is a smallest index $j > i$ such that $(q_{j-1}, \alpha(j-1), q_j) \in F_{h_i(\pi)}$. Now, as π is not accepting, q_i must henceforth be in the label of a child of π , which contradicts the assumption that infinitely many nodes in ϑ have length $|\pi|$.

Thus, π is eventually always stable and always eventually accepting. $\square$

The lemmas in this appendix imply Theorem 2.

B Proof of Lemma 3

Proof. To show (1), we have to run through the properties of a leaf SCC. First, all states in $[S]$ are reachable and connected by the previous lemma. It remains to show that no state $[(m', d')] \notin [S]$ is the successor of any state in the quotient MC. Let us assume for contradiction that there is a state $(m, d) \in S$ such that $[(m', d')]$ is the successor of $[(m, d)]$. By the previous lemma, this implies that there is a state (m', d'') with $\text{reach}(d'') = \text{reach}(d')$ reachable from (m, d) . As S is a leaf SCC, $(m', d'') \in S$ holds, and $[(m', d')] = [(m', d'')] \in [S]$ follows, which is a contradiction.

To show (2), let us start with the set $S'' = \{s \in V \mid [s] \in S\}$. $[S''] = S$ follows from the point (2) of Lemma 2, and S'' is closed under successors: assuming for contradiction that this is not the case provides a $(m, d) \in S''$ with successor $(m', d') \notin S''$. (Note that the reachability of (m, d) implies the reachability of (m', d') .) But this implies that $[(m', d')]$ is a successor of $[(m, d)]$, and by construction of S'' , $[(m', d')] \in S$. For the leaf SCC S , the construction of S'' then implies $(m', d') \in S''$, which is a contradiction.

As S'' is closed under successors, it contains some leaf SCC, and we select S' to be such a leaf SCC. We have shown that $[S']$ is a leaf SCC in the quotient MC in the first half of this proof. Consequently, $[S']$ is a leaf SCC that is contained in S , and hence $[S'] = S$ holds. $\square$

C Proof of Theorem 3

In this appendix, we show Theorem 3, that is, the inclusions $\mathcal{L}(S_{[d]}^u) \subseteq \mathcal{L}(\mathcal{B}_{\langle d \rangle}^u) \subseteq \mathcal{L}(\mathcal{A}_d) \subseteq \mathcal{L}(\mathcal{B}_{\langle d \rangle}^o), \mathcal{L}(S_{[d]}^o)$. The subset automata S^u, S^o , and the breakpoint automata $\mathcal{B}^u, \mathcal{B}^o$ are defined in Section 3.3.

Lemma 8. $\mathcal{L}(S_{[d]}^u) \subseteq \mathcal{L}(\mathcal{B}_{\langle d \rangle}^u)$.

Proof. Let us assume for contradiction that α is an infinite word such that the run ρ of $S_{[d]}^u$ on α is accepting, while the run ρ' of $\mathcal{B}_{\langle d \rangle}^u$ on α is rejecting. Then there is a position $p \in \omega$ such that the following statements hold:

- no transition after position p is accepting in $\bar{\rho}'$, and, consequently,
- there is an index i such that, in the run ρ' of $\mathcal{B}_{\langle d \rangle}^u$, for all $n \geq p$, $\text{index}((\rho'(n))) = i$.

It is easy to see that, by construction, $\text{set}((\rho'(n))) = \rho(n)$ holds for all $n \in \omega$. As ρ is accepting, there is a position $n \geq p$ such that, for all $q \in \rho(n)$ and all $q' \in \rho(n+1)$, $(q, \alpha(n), q') \in T'$ implies $(q, \alpha(n), q') \in F_i$. (Otherwise F_i would not be accepting in $\mathcal{S}_{[d]}^u$.) But then, transition n in $\bar{\rho}'$ is accepting, which is a contradiction. $\square$

Lemma 9. $\mathcal{L}(\mathcal{B}_{\langle d \rangle}^u) \subseteq \mathcal{L}(\mathcal{A}_d)$

Proof. Let us consider an accepting run ρ of $\mathcal{B}_{\langle d \rangle}^u$ and a run ρ' of $\mathcal{A}_d$ on a given input word α . It is easy to show by induction that, for $\rho'(n) = (\mathcal{T}_n, l_n, h_n)$,

- $\text{set}((\rho(n))) = \text{reach}((\rho'(n)))$,
- $\text{children}((\rho(n))) = \bigcup_{i \in \mathcal{T}_n \cap \mathbb{N}} l_n(i)$, and
- $\text{index}((\rho(n))) = h_n(\varepsilon)$

hold for all $n \in \omega$, and that if $\bar{\rho}(n)$ is accepting, then $\bar{\rho}'(n)$ is accepting with accepting pair with index ε . As the root cannot be rejecting, this implies that ρ' is accepting, too. $\square$

Lemma 10. $\mathcal{L}(\mathcal{A}_d) \subseteq \mathcal{L}(\mathcal{B}_{\langle d \rangle}^o)$

Proof. Let us consider an accepting run ρ' of $\mathcal{A}_d$ and a run ρ of $\mathcal{B}_{\langle d \rangle}^o$ on a given input word α . It is easy to show by induction that, for $\rho'(n) = (\mathcal{T}_n, l_n, h_n)$,

- $\text{set}((\rho(n))) = \text{reach}((\rho'(n)))$,
- $\text{children}((\rho(n))) = \bigcup_{i \in \mathcal{T}_n \cap \mathbb{N}} l_n(i)$, and
- $\text{index}((\rho(n))) = h_n(\varepsilon)$

hold for all $n \in \omega$, and that if $\bar{\rho}'(n)$ is accepting with accepting pair with index ε (recall that the root cannot be rejecting), then $\bar{\rho}(n)$ is accepting. If $\bar{\rho}'(n)$ is accepting, but not with index ε , then the node with position 1 in the history is eventually always stable (note that, whenever 1 is not stable, no other node than ε is) say from position $o \in \omega$ onwards. But then $\text{children}((\rho(n))) \supseteq l_n(1)$ cannot be empty for any $n \geq o$. Moreover, the stability of node 1 implies $(l_n(1), \sigma, l_{n+1}(1)) \in T'$, and thus $\bar{\rho}(n) \notin R_1$.

Consequently, $\bar{\rho}$ is rejecting only finitely many times, and α is accepted by $\mathcal{B}_{\langle d \rangle}^o$. $\square$

Lemma 11. $\mathcal{L}(\mathcal{A}_d) \subseteq \mathcal{L}(\mathcal{S}_{[d]}^o)$.

Proof. We use $\mathcal{L}(\mathcal{A}_d) = \mathcal{L}(\mathcal{B}_{[d]})$ (Theorem 2), where $\mathcal{B}$ is the GTBA with $\mathcal{A} = \det(\mathcal{B})$. Let ρ be the run of a word α that is rejected by $\mathcal{S}_{[d]}^o$, and i an index such that $\bar{\rho}$ contains only finitely many transitions in F_i . Then there is a position $n \in \omega$ such that no transition in F_i may occur from n onwards.

But then there is, there are no $m \geq n$ for which a position $q \in \rho(m)$ is reachable such that $(q, \alpha(n), q')$ is in the set F_i of the GTBA $\mathcal{B}$. Thus no run of $\mathcal{B}$ on α can have more than n transitions from F_i , and the claim follows with $\mathcal{L}(\mathcal{A}_d) = \mathcal{L}(\mathcal{B}_{[d]})$ (Theorem 2). $\square$

Together, the lemmas in this appendix imply Theorem 3.

D Markov Decision Processes

In this section, we give more details on how we evaluate MDPs. Let $\mathcal{M} = (M, \Sigma, \mu_0, Steps)$ be an MDP. We recall the computation of the probability $\sup_{\delta} \text{Prob}^{\delta}(\mathcal{B}_{\varphi})$ where $\delta: M \rightarrow \Sigma$ is deterministic scheduler. See [BK08, Lemma 10.125] for details. Let $\mathcal{A} = \text{det}(\mathcal{B}_{\varphi})$ denote the deterministic Rabin automaton for $\mathcal{B}_{\varphi}$.

D.1 Product of MDPs with automata

As for MC, we have to define products of MDP with deterministic automata.

Definition 2 (RMDP and GMDP). *Given an MDP $\mathcal{M} = (M, L, \mu, Steps)$ and a deterministic automaton $\mathcal{D} = (\Sigma, Q, q_0, \delta, ACC)$, the product automaton is defined by $\mathcal{M} \times \mathcal{D} := (M \times Q, L', \mu', Steps', ACC')$ where*

- $Steps'((m, d), a)(m', d')$ equals $Steps(m, a)(m')$ if $d' = \delta(d, L(m'))$, and is 0 otherwise;
- $L'((m, d)) = L(m)$;
- $\mu'((m, d)) = \mu(m)$ if $d = \delta(q_0, L(m))$, and 0 otherwise;
- if $ACC = \{F_i \mid i \in [1..k]\}$, then $ACC' := \{F'_i \mid i \in [1..k]\}$ where $F'_i = \{((m, d), a, (m', d')) \mid (d, L(m'), d') \in F_i, a \in \Sigma\}$ (Generalised Büchi Markov Decision Process, GMDP);
- if $ACC = \{(A_i, R_i) \mid i \in [1..k]\}$, then $ACC' := \{(A'_i, R'_i) \mid i \in [1..k]\}$ where $A'_i = \{((m, d), (m', d')) \mid (d, L(m'), d') \in A_i\}$ and $R'_i = \{((m, d), (m', d')) \mid (d, L(m'), d') \in R_i\}$ (Rabin Markov Decision Process, RMDP).

D.2 Reduction to Probabilistic Reachability

For the reader's convenience, we recall the reduction of $\sup_{\delta} \text{Prob}^{\mathcal{M}, \delta}(\mathcal{B}_{\varphi})$ to probabilistic reachability in the RMDP $\mathcal{M} \times \mathcal{A}$.

First, we need some notations:

- A sub-MDP is a pair (T, en) such that $\emptyset \neq T \subsetneq M$ and $en: M \rightarrow 2^{\Sigma}$ satisfying:
 - (i) $\emptyset \neq en(s) \subseteq Act(s)$ where $Act(s)$ denote the enabled actions of s , (ii) $s \in T$ and $\alpha \in en(s)$ implies $Steps(s, \alpha)$ is a distribution over T .
- An *end component* of $\mathcal{M}$ is a sub-MDP (T, en) such that the digraph induced by (T, en) is strongly connected.
- An end component (T, en) is a *maximal end component* (MEC) if it is not contained in some other end component $(T', en') \neq (T, en)$ with $T \subseteq T'$ and $en(s) \subseteq en'(s)$ for all $s \in T$.

We first define a sub-MDP restricting to a subset of states $B \subseteq M$. Let $\mathcal{M}_{\square B}$ denote the sub-MDP with state space B , moreover, only those transitions that will remain in B with probability 1 will be kept, other transitions will be removed. As we allow absorbing states, we do not need the fresh state *no* as in [BK08].

With these notations, we recall that the computation of the probability $\sup_{\delta} \text{Prob}^{\delta}(\mathcal{B}_{\varphi})$ can be reduced to a reachability probability:

- for a Rabin pair (A, R) , define $U_{(A, R)} \subseteq M$ such that $m \in U_{(A, R)}$ iff there exists a MEC (T, en) in $\mathcal{M}_{\square \overline{R}}$ such that $T \cap A \neq \emptyset$. In this case we say s is accepting w.r.t. (A, R) .

– Let $U := \bigcup U_{(A_i, R_i)}$. Then, $\sup_{\delta} \text{Prob}^{\mathcal{M}, \delta}(\mathcal{B}_{\varphi})$ reduces to a probabilistic reachability:

$$\sup_{\delta} \text{Prob}^{\mathcal{M}, \delta}(\mathcal{B}_{\varphi}) = \sup_{\delta} \text{Prob}^{\mathcal{M} \times \mathcal{A}_{\varphi}, \delta}(\Diamond U)$$

Obviously, if a MEC (T, en) contains an accepting state $s \in U$, all states in T have value 1.

D.3 The incremental evaluation of MECs

As argued in the body of the paper, the likelihood $\sup_{\delta} \text{Prob}_{(m, d)}^{\mathcal{M} \times \mathcal{A}, \delta}(\mathcal{B}_{\varphi})$ does not depend on d itself, but only on $[d]$. Together with the observations from Corollary 6, we can therefore follow the principle layered approach. As for Markov chains, we can use the results from the previous layers to avoid parts of the construction. The incremental evaluation is described as follows:

Construct the quotient MDP $\mathcal{M} \times \mathcal{S}^u$ (which is the same as $\mathcal{M} \times \mathcal{S}^o$ except the accepting conditions). For each MEC C , evaluate as follows:

1. Similar to Lemma 5 for Markov chains, C is accepting if C contains some accepting transition from F_i all $i \in [1..k]$; C is non-accepting if C does not contain states from F_i holds for some $i \in [1..k]$.
(Strictly speaking, the F_i refer to transitions in the automata. They naturally extend to the transitions in the product, and are used in this sense here.)
2. If we could neither establish that C is accepting nor that C is non-accepting, we refine C by a breakpoint construction (only for this MEC). Let $(\mathcal{M} \times \mathcal{B}^u)|_C, (\mathcal{M} \times \mathcal{B}^o)|_C$ denote the breakpoint automata for C (again, the difference is only reflected by the accepting conditions).
These RMDPs are to be read as the MDPs that exist if one expands the MDP restricted to S . Let C' be an arbitrary MEC in the resulting breakpoint automata.
3. C' is accepting if C' contains some accepting transition in A_{ε} . As for all RMDPs, $C' = (T, en)$ is non-accepting if there is no MEC in $(T, en \setminus R_1)$.
(Strictly speaking, A_{ε} and R_1 refer to transitions in the automata. They naturally extend to the transitions in the product, and are used in this sense here.)
4. If C remains inconclusive, we finally evaluate the MECs of $\mathcal{M} \times \mathcal{A}$. For this, we follow a similar procedure as for the breakpoints. In particular, it suffices to refine the MECs individually.

After the above procedure, we make accepting MECs in the quotient MDP absorbing, whereas non-accepting MECs are untouched. We remark that a MEC cannot be accepting if it contains a state $(m, \langle d \rangle)$ such that $\sup_{\delta} \text{Prob}_{(m, d)}^{\mathcal{M} \times \mathcal{A}, \delta}(\mathcal{B}_{\varphi}) < 1$ has been established by a previous estimation.

As a final step, a probabilistic reachability analysis – through solving a linear programming problem – is needed for the evaluation. Note that this can always be performed on the quotient MDP.

E Illustrational examples

E.1 Acceptance conditions

In Fig. 1 we give an (intentionally minimal) example to illustrate the different determination methods and their differences. The starting point is the Büchi automaton $\mathcal{B}_1$ in

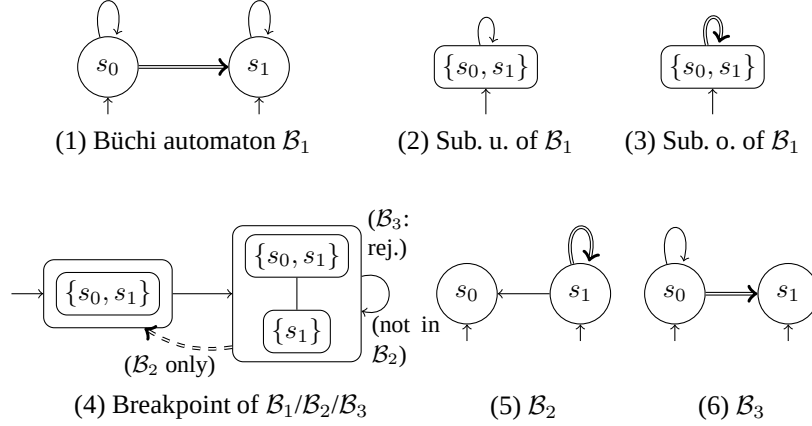


Fig. 1. Example illustrating determinisation approaches.

part (1). We assume a single input character a , so that all transitions in the figure are labelled with a . In addition, there is only one acceptance condition, and we use double arrows for accepting transitions. It is easy to see that the language of the automaton is empty, because the only accepting transition from s_0 to s_1 can be taken at most once.

To determinise the automaton, we first compute under and overapproximating subset automata in (2) and (3). It turns out that the result is inconclusive, because the automaton in (2) accepts the empty language whereas (3) accepts a^* . By applying the breakpoint construction, we obtain the automaton in (4). The result is inconclusive, too. The self-loop in the rightmost state is neither accepting nor rejecting. Thus, using the underapproximating condition, the automaton does not accept any word, while for overapproximating automata the automaton accepts a^* . If we would have applied the Rabin construction, the automaton would be basically the same. Then, all R_v but also all A_v would be empty, so that no word would be accepted.

Assume now that our starting point is $\mathcal{B}_2$ of (5). The language of this automaton is a^* . The breakpoint automaton constructed is basically the same as before in (4), except that instead of the self-loop, we have an accepting transition back to the initial state. This means that the breakpoint construction is conclusive, as both, the under- and the overapproximation, lead to the same language a^* . Now assume that we want to determinise $\mathcal{B}_3$ of (6). The result is similar to the one for $\mathcal{B}_1$. However, the self-loop transition is now rejecting. Thus, the result is conclusive, as the language from both, the under- and the overapproximation, are empty.

E.2 Accepting and rejecting Rabin ECCs in subset ECCs

In the following, we demonstrate that for MDPs, an ECC from the subset construction may lead to both accepting and rejecting ECCs when applying the Rabin construction to decide it. For a subset ECC to be accepting, it suffices to lead to at least one accepting ECC. Thus, it does not suffice to check for a single Rabin ECC if it is rejecting. In the worst case, all Rabin ECCs have to be tried out.

Consider the MDP $\mathcal{M}$ in Figure 2 (1) and the Büchi automaton $\mathcal{B}$ in (2). When we transform $\mathcal{B}$ into subset automata and build the composition with $\mathcal{M}$, we end up with

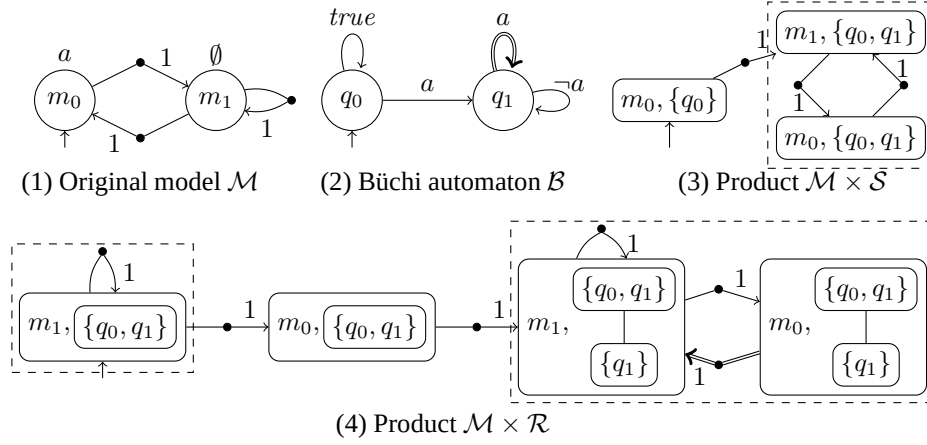


Fig. 2. Example illustrating Subsection E.2.

the composition shown in (3). The product construction is inconclusive, and contains one maximal ECC marked with a dashed box. To decide it, in (4) we build the product with the Rabin determinisation automaton $\mathcal{R}$ of $\mathcal{B}$, starting with one state of the maximal ECC of (3). In (4), there are two ECCs marked with dashed boxes. The one consisting of a single state is rejecting, whereas the one consisting of two states is accepting. Thus, we can only mark the ECC of (3) as accepting after we have considered the accepting ECC of (4). (The transition is accepting for node 1 of the history tree.)

Note that the state we start with is arbitrary. If we had started with the other state from the ECC of (3), then we had obtained a smaller expanded product that does not contain the leftmost state or (4).

E.3 Staying within the ECC is important

When evaluating a maximal ECC of a product of a model and a subset automaton using a determinisation (or breakpoint) construction, it is important to only construct the determinisation for the states within the ECC, leaving aside the states outside the ECC which might be reached. (Note that this will have to be done for each maximal ECC individually.)

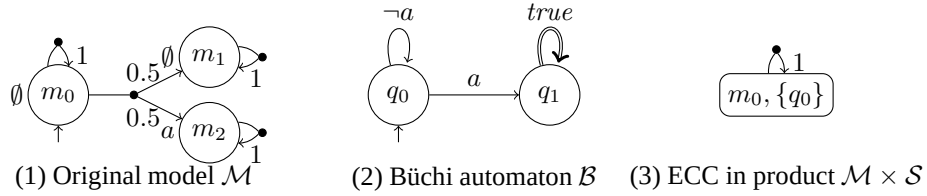


Fig. 3. Example illustrating Subsection E.3.

Consider the model $\mathcal{M}$ of Figure 3 (1) and the Büchi automaton $\mathcal{B}$ in (2). If we build the product of $\mathcal{M}$ with a subset automaton $\mathcal{S}$ of $\mathcal{B}$, the product $\mathcal{M} \times \mathcal{S}$ has the same structure as the model itself. Thus, in the following, we can identify states of $\mathcal{M}$ and $\mathcal{M} \times \mathcal{S}$. It consists of three maximal endcomponents, each consisting of one of $m_0, m_1,$

and m_2 . Assume we want to decide the maximal ECC $\{m_0\}$. If we do so correctly, and restrict to the states of the ECC under consideration, we obtain the ECC of (3). However, if we just use m_0 as the starting point of the determinisation, we end up with a model with the same structure as (1). We would then by mistake identify $\{m_0\}$ as accepting, because $\{m_2\}$ is.