MoChiBA: Probabilistic LTL Model Checking using limit-deterministic Büchi Automata

Salomon Sickert and Jan Křetínský

Technische Universität München

Abstract. The limiting factor for quantitative analysis of Markov decision processes (MDP) against specifications given in linear temporal logic (LTL) is the size of the generated product. As recently shown, a special subclass of limit-deterministic Büchi automata (LDBA) can replace deterministic Rabin automata in quantitative probabilistic model checking algorithms. We present an extension of PRISM for LTL model checking of MDP using LDBA. While existing algorithms can be used only with minimal changes, the new approach takes advantage of the special structure and the smaller size of the obtained LDBA to speed up the model checking. We demonstrate the speed up experimentally by a comparison with other approaches.

1 Introduction

Linear temporal logic (LTL) [30] is a prominent specification language and has been proven useful in industrial practice. The key to efficient LTL model checking is the automata-theoretic approach [38]: first, a given LTL formula is translated into an automaton; second, a product of the system and the automaton is constructed and analysed. Since real systems are huge, it is crucial to construct *small automata* in order to avoid a large size increase of the product.

LTL is typically translated into non-deterministic Büchi automata (NBA) [8, 10, 14, 36, 17, 18, 15, 2, 11]. However, for probabilistic models such as Markov decision processes (MDP) non-deterministic automata are not applicable [3] and the standard solution is to *determinise* them using Safra's construction [32, 29, 33, 22, 37]. This approach is implemented in the most widespread probabilistic model checker PRISM [27]. However, the determinisation step is costly and often increases the size of automata dramatically. Therefore, *direct translations of LTL to deterministic automata* have been proposed [26, 25, 1, 13], implemented [16, 5, 24], and shown to be more efficient for probabilistic model checking [6]. Nevertheless, despite more sophisticated acceptance conditions, such as generalized Rabin [26], the imposed determinism inevitably increases the size of the automata.

This naturally raises the question whether fully deterministic automata are necessary or whether restricted forms of determinism are sufficient. For instance, in the setting of games where NBA are not applicable either, a weaker notion of determinism called *good-for-games automata* is sufficient [20]. It has been proven sufficient also for probabilistic model checking, but practically "did not improve on the standard approach" [23]. Further, *unambiguous automata* can be

used [9] for model checking Markov chains, but not for MDPs. Moreover, *limit-deterministic* Büchi automata (LDBA) [38, 7] can be used for probabilistic model checking MDPs in the qualitative case (deciding whether a property holds with probability 1). This idea has been further explored also in the quantitative setting (computing the probability of satisfaction) and an algorithm constructing products with several (limit-)deterministic automata proposed [19]. Although LDBA cannot in general be used for probabilistic model checking, a recent translation [35] of LTL produces LDBA, which can be used in the standard algorithm based on the construction of a single product. It also discusses the subclass of LDBA that can be used for this task. Note that there is also an exponentially better translation [21] (based on [25]) of a fragment of LTL called $\mathrm{LTL}_{\setminus \mathbf{GU}}$ into LDBA and that there is also an efficient complementation procedure for LDBA [4].

In this paper, we provide the first implementation of the probabilistic model checking procedure proposed in [35] based on LDBA. Apart from smaller sizes of LDBA, another advantage of the Büchi acceptance condition is a faster analysis of maximal end components (MECs), compared to the standard repetitive recomputation for each Rabin pair. We also present several crucial optimizations, which make our implementation outperform other approaches on many formulas. We illustrate this on experimental results. The tool as well as the explanation of its name can be found on `https://www7.in.tum.de/~sickert/projects/mochiba/`.

2 Overview of the Algorithm

In order to present our implementation and optimizations, we have to sketch how an MDP $\mathcal{M}$ is checked against an LTL formula φ by the algorithm of [35]. First, φ is translated into an LDBA $\mathcal{A}(\varphi)$. Second, $\mathcal{M}$ is checked against $\mathcal{A}(\varphi)$ using a straightforward extension of the standard algorithm.

LDBA Construction. An LDBA is a (possibly generalised) Büchi automaton partitioned into an *initial* and an *accepting* part, where the initial part contains no accepting transitions and the accepting part is deterministic. Moreover, the construction of [35] produces LDBA with the initial part deterministic except for ε-transitions into the accepting part.

We illustrate the translation on $\varphi = a \wedge \mathbf{X}(\mathbf{FG}a \vee \mathbf{FG}b)$. Each state in the initial part is labeled with a formula. The words accepted from a state are exactly those satisfying the formula. Observe that $\mathbf{FG}a \vee \mathbf{FG}b$ holds iff eventually we reach a point where $\mathbf{G}a$ holds or $\mathbf{G}b$ holds. We non-deterministically guess this point and take the ε-transition to the accepting part, where we check the guess.

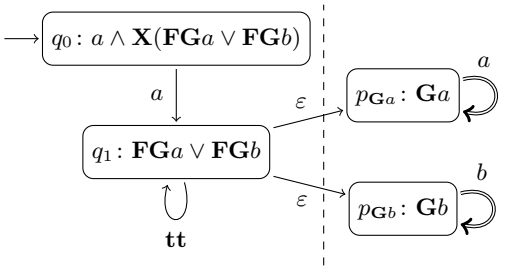

Fig. 1. LDBA $\mathcal{A}(\varphi)$ with the initial part on the left and the accepting on the right.

For this formula `spot` (2.0) produces a deterministic Rabin automaton with 4 states, too, but adding two more disjuncts $\mathbf{FG}c$ and $\mathbf{FG}d$ increases the size to 26 states. In contrast, our LDBA requires only two extra states.

Product Construction and Analysis. We proceed according to the standard algorithm:

1. Construct the product $\mathcal{P} = \mathcal{M} \times \mathcal{A}(\varphi)$.
2. Compute maximal end-components (MECs) of $\mathcal{P}$.
3. Compute the maximum probability to reach winning MECs. A MEC is winning if it satisfies the acceptance condition of $\mathcal{A}(\varphi)$: here, if it contains an accepting transition for each Büchi condition.

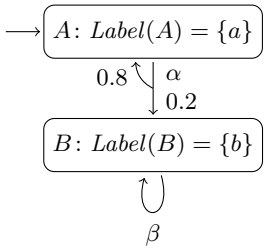

Fig. 2. An MDP $\mathcal{M}$.

The standard product of an MDP and a deterministic automaton defines the transitions (in the usual notation) by $P(\langle s, q\rangle, \alpha, \langle s', q'\rangle) = P(s, \alpha, s')$ if $q' = \delta(q, \mathit{Label}(s'))$ and otherwise equals 0. We extend the procedure to handle also non-deterministic ε-transitions by additional actions: let $q_1, \ldots, q_n$ be the successors of q under ε, then for each $i = 1, \ldots, n$ we add a new action called ε_{q_i} and define $P(\langle s, q\rangle, \varepsilon_{q_i}, \langle s, q_i\rangle) = 1$ (note that s does not move here).

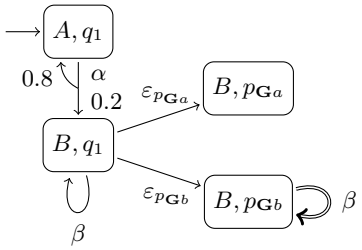

Fig. 3. The product $\mathcal{M} \times \mathcal{A}(\varphi)$.

Fig. 3 illustrates the construction by a product of the system of Fig. 2 and the automaton of Fig. 1. A crucial optimization used here is that it is sufficient to take ε-transitions only from states in MECs of $\mathcal{M} \times \mathcal{N}(\varphi)$ (which are exactly MECs of the product of $\mathcal{M}$ and the initial part of $\mathcal{A}(\varphi)$). Hence no ε-transitions have to be produced in the initial state here.

3 Implementation and Optimizations

`MoChiBA` [34] replaces the LTL model-checker and the MEC computation in the explicit-state model-checker of PRISM, while other infrastructure (parsing, model construction, probability computation) are inherited from PRISM. The tool cannot be configured — all optimizations are enabled — and does not need to be installed. It reads a model (given as an MDP, `.nm`) and a property specification (`.pctl`) and prints the results to `stdout`:

```
./mochiba.sh model.nm properties.pctl
```

Apart from taking ε-transitions only from states in MECs of $\mathcal{M} \times \mathcal{N}(\varphi)$ as mentioned above, we implement the following optimizations:
Transition-based acceptance leads to smaller automata, compared to state-based acceptance. Consequently, it is used by many translators, for instance [11,

1, 24]. However, PRISM translates all automata to state-based, thus increasing the size of the product. Our procedure avoids this and constructs and analyses directly the transition-based product.

Generalised Büchi acceptance condition allows for more efficient analysis than (generalised) Rabin, Streett, or parity conditions. Indeed, for the latter conditions expensive re-computations of MECs are necessary to handle different sets to be visited finitely often. In contrast, we compute MECs only once and check whether each set to be visited infinitely often intersects the MEC.

A single trap state is present in the product. Should the product enter any state from which the automaton component can never accept, the exploration of this part stops and redirects the transition to the single trap state.

Primitive data structures such as arrays are used instead of the more flexible Java collections, since they are more memory efficient, as boxing into objects is not necessary.

Sparse bit sets have proven more memory efficient for our approach than plain bit sets with a mapping table.

MEC decomposition is performed locally on disconnected accepting parts (corresponding to different ε-transitions). Together with the use of sparse bit sets, MECs are computed faster and using less memory.

4 Experimental Evaluation

We evaluate our novel approach in the setting of [6, 19]: we consider the Pneuli-Zuck randomised mutual exclusion protocol [31] of the PRISM benchmark suite [28] and also the same previously considered formulas (see lines 1–10 of Table 1). Additionally, lines 11–14 consider the deeply nested formulas of [35]. Finally, complementary to the **GF**-, **FG**- and fairness-like properties, lines 15–16 include simple reachability properties, which lie in the focus of the traditional methods.

The experiments were performed on a 2.5 GHz Intel Core i7 (I7-4870HQ) and granted 12 GB RAM and 1 hour computing time for model checking each property (given the model already in the memory). We denote time-outs and mem-outs by "-". We compare the following tools

- `MoChiBA` (1.0) [34] is our implementation based on the LDBA translation of [35] and the explicit model checker of PRISM.
- `PRISM` (4.3) [27] with the symbolic engine, which is the fastest here, and with the following translators:
  - Built-in LTL to deterministic Rabin automaton translation, re-implementing ltl2dstar [22].
  - `Rabinizer` (3.1) [24] using the Safra-less direct translation into generalised Rabin automata, which are now supported by PRISM.
- `IscasMC` (unofficial, unversioned) implements the lazy approach of [19], using SPOT 1.2.6 [12] to translate LTL to non-deterministic Büchi automaton. We used the two fastest configurations as listed in [19]:
  - Multi-breakpoint (`BP`) construction with the explicit engine.
  - Rabin (`R`) construction with the explicit engine.

Table 1. Runtime comparison on model checking these properties on the Pneuli-Zuck randomised mutual exclusion protocol [31].

property	n	time (rounded, in seconds) MoChiBA	PRISM	Rabinizer	IscasMC-BP	IscasMC-R
(1) $\mathbb{P}_{max=?}[\substack{\mathbf{GF}p1=10 \wedge \mathbf{GF}p2=10 \\ \wedge \mathbf{GF}p3=10}]$	4	$< \mathbf{1}$	16	$< \mathbf{1}$	$< \mathbf{1}$	$< \mathbf{1}$
	5	2	230	$< \mathbf{1}$	12	11
(2) $\mathbb{P}_{max=?}[\substack{\mathbf{GF}p1=10 \wedge \mathbf{GF}p2=10 \\ \wedge \mathbf{GF}p3=10 \wedge \mathbf{GF}p4=10}]$	4	$< \mathbf{1}$	26	$< \mathbf{1}$	1	$< \mathbf{1}$
	5	2	345	$< \mathbf{1}$	12	12
(3) $\mathbb{P}_{min=?}[\substack{\mathbf{GF}p1=10 \wedge \mathbf{GF}p2=10 \\ \wedge \mathbf{GF}p3=10 \wedge \mathbf{GF}p4=10}]$	4	**1**	3552	33	**1**	22
	5	**11**	-	572	18	641
(4) $\mathbb{P}_{max=?}[\substack{(\mathbf{GF}p1=0 \vee \mathbf{FG}p2\neq 0) \\ \wedge (\mathbf{GF}p2=0 \vee \mathbf{FG}p3\neq 0)}]$	4	**1**	684	18	2	4
	5	**15**	-	293	19	50
(5) $\mathbb{P}_{max=?}[\substack{(\mathbf{GF}p1=0 \vee \mathbf{FG}p1\neq 0) \\ \wedge (\mathbf{GF}p2=0 \vee \mathbf{FG}p2\neq 0)}]$	4	$< \mathbf{1}$	$< \mathbf{1}$	23	1	4
	5	1	$< \mathbf{1}$	403	17	59
(6) $\mathbb{P}_{max=?}[\substack{(\mathbf{GF}p1=0 \vee \mathbf{FG}p2\neq 0) \\ \wedge (\mathbf{GF}p2=0 \vee \mathbf{FG}p3\neq 0) \\ \wedge (\mathbf{GF}p3=0 \vee \mathbf{FG}p1\neq 0)}]$	4	$< \mathbf{1}$	78	9	3	10
	5	**10**	1293	137	29	143
(7) $\mathbb{P}_{max=?}[\substack{(\mathbf{GF}p1=0 \vee \mathbf{FG}p1\neq 0) \\ \wedge (\mathbf{GF}p2=0 \vee \mathbf{FG}p2\neq 0) \\ \wedge (\mathbf{GF}p3=0 \vee \mathbf{FG}p3\neq 0)}]$	4	$< \mathbf{1}$	$< \mathbf{1}$	61	2	18
	5	1	$< \mathbf{1}$	1077	27	277
(8) $\mathbb{P}_{min=?}[\substack{(\mathbf{GF}p1\neq 10 \vee \mathbf{GF}p1=0 \vee \mathbf{FG}p1=1) \\ \wedge \mathbf{GF}p1\neq 0 \wedge \mathbf{GF}p1=1}]$	4	$< \mathbf{1}$	8	8	1	1
	5	**1**	145	190	16	21
(9) $\mathbb{P}_{max=?}[\substack{(\mathbf{G}p1\neq 10 \vee \mathbf{G}p2\neq 10 \vee \mathbf{G}p3\neq 10) \\ \wedge (\mathbf{FG}p1\neq 1 \vee \mathbf{GF}p2=1 \vee \mathbf{GF}p3=1) \\ \wedge (\mathbf{FG}p2\neq 1) \vee \mathbf{GF}p1=1 \vee \mathbf{GF}p3=1)}]$	4	**5**	-	1195	8	871
	5	**99**	-	-	125	-
(10) $\mathbb{P}_{min=?}[\substack{\mathbf{FG}p1\neq 0 \vee \mathbf{FG}p2\neq 0 \\ \vee \mathbf{GF}p3=0 \vee (\mathbf{FG}p1\neq 10 \\ \wedge \mathbf{GF}p2=10 \wedge \mathbf{GF}p3=10)}]$	4	**1**	728	33	79	6
	5	**24**	-	486	-	77
(11) $\mathbb{P}_{min=?}[f_{0,0}] = \mathbb{P}_{min=?}[\substack{(\mathbf{GF}p1=10)\mathbf{U} \\ (p2=10)}]$	4	$< \mathbf{1}$	17	40	2	2
	5	**11**	257	715	23	54
(12) $\mathbb{P}_{max=?}[f_{0,4}] = \mathbb{P}_{max=?}[\substack{(\mathbf{GF}p1=10)\mathbf{U} \\ (\mathbf{XXXX}p2=10)}]$	4	$< \mathbf{1}$	3	$< \mathbf{1}$	1	15
	5	5	20	**2**	20	2381
(13) $\mathbb{P}_{min=?}[f_{1,0}] = \mathbb{P}_{min=?}[\substack{(\mathbf{GF}p1=10)\mathbf{U} \\ (\mathbf{G}((\mathbf{GF}p2=10)\mathbf{U} \\ (p3=10)))}]$	4	$< \mathbf{1}$	909	22	314	4
	5	**13**	-	436	-	59
(14) $\mathbb{P}_{max=?}[f_{1,4}] = \mathbb{P}_{max=?}[\substack{(\mathbf{GF}p1=10)\mathbf{U} \\ (\mathbf{G}((\mathbf{GF}p2=10)\mathbf{U} \\ (\mathbf{XXXX}p3=10)))}]$	4	$< \mathbf{1}$	-	18	2	2
	5	**12**	-	285	24	25
(15) $\mathbb{P}_{max=?}[p1 = 0 \ \ \mathbf{U} \ \ p2 = 10]$	4	$< \mathbf{1}$	$< \mathbf{1}$	$< \mathbf{1}$	$< \mathbf{1}$	$< \mathbf{1}$
	5	$< \mathbf{1}$	$< \mathbf{1}$	$< \mathbf{1}$	7	7
(16) $\mathbb{P}_{max=?}[\mathbf{XXXXXX}p1 = 0]$	4	$< \mathbf{1}$	$< \mathbf{1}$	$< \mathbf{1}$	1	$< \mathbf{1}$
	5	3	$< \mathbf{1}$	$< \mathbf{1}$	19	16

5 Conclusion

We have implemented a novel approach for probabilistic LTL model checking using a subclass of non-deterministic Büchi automata. Since the experimental results for the explicit state-space implementation are encouraging, we plan to extend the approach to a symbolic one. Further, a parellelisation of the product construction and MECs analysis, as well as dedicated constructions for the *Release*-operator or various LTL fragments could lead to further speed ups.

Acknowledgements. This work is partially funded by the DFG Research Training Group "PUMA: Programm- und Modell-Analyse" (GRK 1480) and by the Czech Science Foundation, grant No. 15-17564S.

The authors want to thank Ernst Moritz Hahn and Andrea Turrini for providing a private version of IscasMC to compare to and for assistance in using it.

References

1. Babiak, T., Blahoudek, F., Křetínský, M., Strejček, J.: Effective translation of LTL to deterministic Rabin automata: Beyond the (F, G)-fragment. In: ATVA. pp. 24–39 (2013)
2. Babiak, T., Křetínský, M., Řehák, V., Strejček, J.: LTL to Büchi automata translation: Fast and more deterministic. In: TACAS. pp. 95–109 (2012)
3. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
4. Blahoudek, F., Heizmann, M., Schewe, S., Strejček, J., Tsai, M.: Complementing semi-deterministic Büchi automata. In: TACAS. LNCS, vol. 9636, pp. 770–787 (2016)
5. Blahoudek, F., Křetínský, M., Strejček, J.: Comparison of LTL to deterministic Rabin automata translators. In: LPAR. LNCS, vol. 8312, pp. 164–172 (2013)
6. Chatterjee, K., Gaiser, A., Křetínský, J.: Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In: CAV. pp. 559–575 (2013)
7. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM 42(4), 857–907 (1995)
8. Couvreur, J.M.: On-the-fly verification of linear temporal logic. In: FM. pp. 253–271 (1999)
9. Couvreur, J., Saheb, N., Sutre, G.: An optimal automata approach to LTL model checking of probabilistic systems. In: LPAR. LNCS, vol. 2850, pp. 361–375 (2003)
10. Daniele, M., Giunchiglia, F., Vardi, M.Y.: Improved automata generation for linear temporal logic. In: CAV. pp. 249–260 (1999)
11. Duret-Lutz, A.: Manipulating LTL formulas using spot 1.0. In: ATVA. pp. 442–445 (2013)
12. Duret-Lutz, A.: LTL translation improvements in Spot 1.0. IJCCBS 5(1/2), 31–54 (Mar 2014)
13. Esparza, J., Křetínský, J.: From LTL to deterministic automata: A Safraless compositional approach. In: CAV. pp. 192–208 (2014)
14. Etessami, K., Holzmann, G.J.: Optimizing Büchi automata. In: CONCUR. pp. 153–167 (2000)
15. Fritz, C.: Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata. In: CIAA. pp. 35–48 (2003)
16. Gaiser, A., Křetínský, J., Esparza, J.: Rabinizer: Small deterministic automata for LTL(F,G). In: ATVA. pp. 72–76 (2012)
17. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: CAV. pp. 53–65 (2001), tool accessible at `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/`
18. Giannakopoulou, D., Lerda, F.: From states to transitions: Improving translation of LTL formulae to Büchi automata. In: FORTE. pp. 308–326 (2002)
19. Hahn, E.M., Li, G., Schewe, S., Turrini, A., Zhang, L.: Lazy probabilistic model checking without determinisation. In: CONCUR. LIPIcs, vol. 42, pp. 354–367 (2015)

20. Henzinger, T.A., Piterman, N.: Solving games without determinization. In: CSL. LNCS, vol. 4207, pp. 395–410. Springer (2006)
21. Kini, D., Viswanathan, M.: Limit deterministic and probabilistic automata for LTL \ GU. In: TACAS. LNCS, vol. 9035, pp. 628–642 (2015)
22. Klein, J.: ltl2dstar - LTL to deterministic Streett and Rabin automata. http://www.ltl2dstar.de/
23. Klein, J., Müller, D., Baier, C., Klüppelholz, S.: Are good-for-games automata good for probabilistic model checking? In: LATA. LNCS, vol. 8370, pp. 453–465 (2014)
24. Komárková, Z., Křetínský, J.: Rabinizer 3: Safraless translation of LTL to small deterministic automata. In: ATVA. LNCS, vol. 8837, pp. 235–241 (2014)
25. Křetínský, J., Ledesma-Garza, R.: Rabinizer 2: Small deterministic automata for $\mathrm{LTL}_{\setminus \mathbf{GU}}$. In: ATVA. pp. 446–450 (2013)
26. Křetínský, J., Esparza, J.: Deterministic automata for the (F,G)-fragment of LTL. In: CAV. pp. 7–22 (2012)
27. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. pp. 585–591 (2011)
28. Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM benchmark suite. In: QEST. pp. 203–204 (2012)
29. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: LICS. pp. 255–264 (2006)
30. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57 (1977)
31. Pnueli, A., Zuck, L.D.: Verification of multiprocess probabilistic protocols. Distributed Computing 1(1), 53–72 (1986), http://dx.doi.org/10.1007/BF01843570
32. Safra, S.: On the complexity of omega-automata. In: FOCS. pp. 319–327 (1988)
33. Schewe, S.: Tighter bounds for the determinisation of Büchi automata. In: FoSSaCS. LNCS, vol. 5504, pp. 167–181 (2009)
34. Sickert, S.: MoChiBA. https://www7.in.tum.de/~sickert/projects/mochiba/
35. Sickert, S., Esparza, J., Jaax, S., Křetínský, J.: Limit-deterministic Büchi automata for linear temporal logic. In: CAV (2016)
36. Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: CAV. pp. 248–263 (2000)
37. Tsai, M.H., Tsay, Y.K., Hwang, Y.S.: GOAL for games, omega-automata, and logics. In: CAV. pp. 883–889 (2013)
38. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: LICS. pp. 332–344 (1986)