Rabinizer 4: From LTL to Your Favourite Deterministic Automaton*

Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler

Technical University of Munich

Abstract. We present Rabinizer 4, a tool set for translating formulae of linear temporal logic to different types of deterministic ω-automata. The tool set implements and optimizes several recent constructions, including the first implementation translating the frequency extension of LTL. Further, we provide a distribution of PRISM that links Rabinizer and offers model checking procedures for probabilistic systems that are not in the official PRISM distribution. Finally, we evaluate the performance and in cases with any previous implementations we show enhancements both in terms of the size of the automata and the computational time, due to algorithmic as well as implementation improvements.

1 Introduction

Automata-theoretic approach [VW86] is a key technique for verification and synthesis of systems with linear-time specifications, such as formulae of linear temporal logic (LTL) [Pnu77]. It proceeds in two steps: first, the formula is translated into a corresponding automaton; second, the product of the system and the automaton is further analyzed. The size of the automaton is important as it directly affects the size of the product and thus largely also the analysis time, particularly for deterministic automata and probabilistic model checking in a very direct proportion. For verification of non-deterministic systems, mostly non-deterministic Büchi automata (NBA) are used [EH00,SB00,GO01,GL02,BKŘS12,DLLF+16] since they are typically very small and easy to produce.

Probabilistic LTL model checking cannot profit directly from NBA. Even the qualitative question, whether a formula holds with probability 0 or 1, requires automata with at least a restricted form of determinism. The prime example are the limit-deterministic (also called semi-deterministic) Büchi automata (LDBA) [CY88] and the generalized LDBA (LDGBA). However, for the general quantitative questions, where the probability of satisfaction is computed, general limit-determinism is not sufficient. Instead, deterministic Rabin automata (DRA) have been mostly used [KNP11] and recently also deterministic generalized Rabin automata (DGRA) [CGK13]. In principle, all standard types of deterministic automata are applicable here except for deterministic Büchi automata (DBA), which are not as expressive as

* This work has been partially supported by the Czech Science Foundation grant No. P202/12/G061 and the German Research Foundation (DFG) project KR 4890/1-1 "Verified Model Checkers" (317422601). A part of the frequency extension has been implemented within Google Summer of Code 2016.

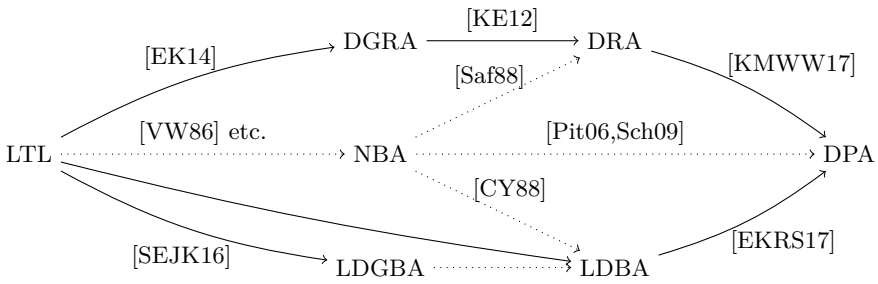

Fig. 1. LTL translations to different types of automata. Translations implemented in Rabinizer 4 are indicated with a solid line. The traditional approaches are depicted as dotted arrows. The determinization of NBA to DRA is implemented in ltl2dstar [Kle], to LDBA in Seminator [BDK+17] and to (mostly) DPA in spot [DLLF+16].

LTL. However, other types of automata, such as deterministic Muller and deterministic parity automata (DPA) are typically larger than DGRA in terms of acceptance condition or the state space, respectively.[1] Recently, several approaches with specific LDBA were proved applicable to the quantitative setting [HLS+15,SEJK16] and competitive with DGRA. Besides, model checking MDP against LTL properties involving frequency operators [BDL12] also allows for an automata-theoretic approach, via deterministic generalized Rabin mean-payoff automata (DGRMA) [FKK15].

LTL synthesis can also be solved using the automata-theoretic approach. Although DRA and DGRA transformed into games can be used here, the algorithms for the resulting Rabin games [PP06] are not very efficient in practice. In contrast, DPA may be larger, but in this setting they are the automata of choice due to the good practical performance of parity-game solvers [FL09,ML16,JBB+17].

Types of translations. The translations of LTL to NBA, e.g., [VW86], are typically *"semantic"* in the sense that each state is given by a set of logical formulae and the language of the state can be captured in terms of semantics of these formulae. In contrast, the determinization of Safra [Saf88] or its improvements [Pit06,Sch09,TD14,FL15] are not "semantic" in the sense that they ignore the structure and produce trees as the new states that, however, lack the logical interpretation. As a result, if we apply Safra's determinization on semantically created NBA, we obtain DRA that lack the structure and, moreover, are unnecessarily large since the construction cannot utilize the original structure. In contrast, the recent works [KE12,KLG13,EK14,KV15,SEJK16,EKRS17,MS17,KV17] provide "semantic" constructions, often producing smaller automata. Furthermore, various transformations such as degeneralization [KE12], index appearance record [KMWW17] or determinization of limit-deterministic automata [EKRS17] preserve the semantic description, allowing for further optimizations of the resulting automata.

Our contribution. While all previous versions of Rabinizer [GKE12,KLG13,KK14] featured only the translation LTL→DGRA→DRA, Rabinizer 4 now implements all the translations depicted by the solid arrows in Fig. 1. It improves all these

[1] Note that every DGRA can be written as a Muller automaton on the same state space with an exponentially-sized acceptance condition, and DPA are a special case of DRA and thus DGRA.

translations, both algorithmically and implementation-wise, and moreover, features the first implementation of the translation of a frequency extension of LTL [FKK15].

Further, in order to utilize the resulting automata for verification, we provide our own distribution[2] of the PRISM model checker [KNP11], which allows for model checking MDP against LTL using not only DRA and DGRA, but also using LDBA and against frequency LTL using DGRMA. Finally, the tool can turn the produced DPA into parity games between the players with input and output variables. Therefore, when linked to parity-game solvers, Rabinizer 4 can be also used for LTL synthesis.

Rabinizer 4 is freely available at `http://rabinizer.model.in.tum.de` together with an on-line demo, visualization, usage instructions and examples.

2 Functionality

We recall that the previous version Rabinizer 3 has the following functionality:

- It translates LTL formulae into equivalent DGRA or DRA.
- It is linked to PRISM, allowing for probabilistic verification using DGRA (previously PRISM could only use DRA).

2.1 Translations

Rabinizer 4 inputs formulae of LTL and outputs automata in the standard HOA format [BBD+15], which is used, e.g., as the input format in PRISM. Automata in the HOA format can be directly visualized, displaying the "semantic" description of the states. Rabinizer 4 features the following command-line tools for the respective translations depicted as the solid arrows in Fig. 1:

ltl2dgra and **ltl2dra** correspond to the original functionality of Rabinizer 3, i.e., they translate LTL (now with the extended syntax, including all common temporal operators) to DGRA and DRA [EK14], respectively.

ltl2ldgba and **ltl2ldba** translate LTL to LDGBA using the construction of [SEJK16] and to LDBA, respectively. The latter is our modification of the former, which produces smaller automata than chaining the former with the standard degeneralization.

ltl2dpa translates LTL to DPA using two modes:

- The default mode uses the translation to LDBA, followed by a LDBA-to-DPA determinization [EKRS17] specially tailored to LDBA with the "semantic" labelling of states, avoiding additional exponential blow-up of the resulting automaton.
- The alternative mode uses the translation to DRA, followed by our improvement of the index appearance record of [KMWW17].

fltl2dgrma translates the frequency extension of $\mathrm{LTL}_{\setminus \mathbf{GU}}$, i.e. $\mathrm{LTL}_{\setminus \mathbf{GU}}$ [KLG13] with $\mathbf{G}^{\sim\rho}$ operator[3], to DGRMA using the construction of [FKK15].

[2] Merging these features into the public release of PRISM as well as linking the new version of Rabinizer is subject to current collaboration with the authors of PRISM.

[3] The *frequential globally* construct [BDL12,BMM14] $\mathbf{G}^{\sim\rho}\varphi$ with $\sim \in \{\geq, >, \leq, <\}, \rho \in [0, 1]$ intuitively means that the fraction of positions satisfying φ satisfies $\sim\rho$. Formally, the fraction on an infinite run is defined using the long-run average [BMM14].

2.2 Verification and synthesis

The resulting automata can be used for model checking probabilistic systems and for LTL synthesis. To this end, we provide our own distribution of the probabilistic model checker PRISM as well as a procedure transforming automata into games to be solved.

Model checking: PRISM distribution For model checking Markov chains and Markov decision processes, PRISM [KNP11] uses DRA and recently also more efficient DGRA [CGK13,KK14]. Our distribution, which links Rabinizer, additionally features model checking using the LDBA [SEJK16,SK16] that are created by our **ltl2ldba**.

Further, the distribution provides an implementation of frequency $\mathrm{LTL}_{\setminus \mathbf{GU}}$ model checking, using DGRMA. To the best of our knowledge, there are no other implemented procedures for logics with frequency. Here, techniques of linear programming for multi-dimensional mean-payoff satisfaction [CKK15] and the model-checking procedure of [FKK15] are implemented and applied.

Synthesis: Games The automata-theoretic approach to LTL synthesis requires to transform the LTL formula into a game of the input and output players. We provide this transformer and thus an end-to-end LTL synthesis solution, provided a respective game solver is linked. Since current solutions to Rabin games are not very efficient we implemented a transformation of DPA into parity games and a serialization to the format of PG Solver [FL09]. Due to the explicit serialization, we foresee the main use in quick prototyping.

3 Optimizations, Implementation, and Evaluation

Compared to the theoretical constructions and previous implementations, there are numerous improvements, heuristics, and engineering enhancements. We evaluate the improvements both in terms of the size of the resulting automaton as well as the running time. When comparing with respect to the original Rabinizer functionality, we compare our implementation **ltl2dgra** to the previous version Rabinizer 3.1, which is already a significantly faster [EKS16] re-implementation of the official release Rabinizer 3 [KK14]. All of the benchmarks have been executed on a host with i7-4700MQ CPU (4x2.4 GHz), running Linux 4.9.0-5-amd64 and the Oracle JRE 9.0.4+11 JVM. Due to the start-up time of JVM, all times below 2 seconds are denoted by <2 and not specified more precisely. All experiments were given a time-out of 900 seconds and mem-out of 4GB, denoted by −.

Algorithmic improvements and heuristics for each of the translations:

ltl2dgra and **ltl2dra** These translations create a master automaton monitoring the satisfaction of the given formula and a dedicated slave automaton for each subformula of the form $\mathbf{G}\psi$ [EK14]. We (i) simplify several classes of slaves and (ii) "suspend" (in the spirit of [BBDL$^+$13]) some so that they appear in the final product only in some states. The effect on the size of the state space is illustrated in Table 1 on a nested formula. Further, (iii) the acceptance condition is considered separately for each strongly connected component (SCC) and then

Table 1. Effect of simplifications and suspension for **ltl2dgra** on the formulae $\psi_i = \mathbf{G}\phi_i$ where $\phi_1 = a_1, \phi(i) = (a_i \mathbf{U}(\mathbf{X}\phi_{i-1}))$, and $\psi'_i = \mathbf{G}\phi'_i$ where $\phi'_1 = a_1$, $\phi'_1 = (\phi'_{i-1}\mathbf{U}(\mathbf{X}^i a_i)$, displaying execution time in seconds / #states.

	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6
Rabinizer 3.1 [EKS16]	<2 / 4	<2 / 16	<2 / 73	3 / 332	60 / 1463
ltl2dgra	<2 / 3	<2 / 7	<2 / 35	3 / 199	13 / 1155
	ψ'_2	ψ'_3	ψ'_4	ψ'_5	ψ'_6
Rabinizer 3.1 [EKS16]	<2 / 4	<2 / 16	2 / 104	128 / 670	–
ltl2dgra	<2 / 3	<2 / 10	<2 / 38	7 / 175	239 / 1330

Table 2. Effect of computing acceptance sets per SCC on formulae $\psi_1 = x_1 \wedge \phi_1$, $\psi_2 = (x_1 \wedge \phi_1) \vee (\neg x_1 \wedge \phi_2)$, $\psi_3 = (x_1 \wedge x_2 \wedge \phi_1) \vee (\neg x_1 \wedge x_2 \wedge \phi_2) \vee (x_1 \wedge \neg x_2 \wedge \phi_3)$, ..., where $\phi_i = \mathbf{XG}((a_i \mathbf{U} b_i) \vee (c_i \mathbf{U} d_i))$, displaying execution time in seconds / #acceptance sets.

	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	...	ψ_8
Rabinizer 3.1 [EKS16]	<2 / 2	<2 / 7	<2 / 19	–	–		–
ltl2dgra	<2 / 1	<2 / 1	<2 / 1	<2 / 1	<2 / 1		<2 / 1

Table 3. Effect of break-point elimination for **ltl2ldba** on safety formulae $s(n,m) = \bigwedge_{i=1}^{n} \mathbf{G}(a_i \vee \mathbf{X}^m b_i)$ and for **ltl2ldgba** on liveness formulae $l(n,m) = \bigwedge_{i=1}^{n} \mathbf{GF}(a_i \wedge \mathbf{X}^m b_i)$, displaying #states (#Büchi conditions)

	$s(1,3)$	$s(2,3)$	$s(3,3)$	$s(4,3)$	$s(1,4)$	$s(2,4)$	$s(3,4)$	$s(4,4)$
[SEJK16]	20 (1)	400 (2)	$8 \cdot 10^3$ (3)	$16 \cdot 10^4$ (4)	48 (1)	2304 (2)	110592 (3)	–
ltl2ldba	8 (1)	64 (1)	512 (1)	4096 (1)	16 (1)	256 (1)	4096 (1)	65536 (1)

	$l(1,1)$	$l(2,1)$	$l(3,1)$	$l(4,1)$	$l(1,4)$	$l(2,4)$	$l(3,4)$	$l(4,4)$
[SEJK16]	3 (1)	9 (2)	27 (3)	81 (4)	10 (1)	100 (2)	10^3 (3)	10^4 (4)
ltl2ldgba	3 (1)	5 (2)	9 (3)	17 (4)	3 (1)	5 (2)	9 (3)	17 (4)

combined. On a concrete example of Table 2, the automaton for $i = 8$ has 31 atomic propositions, whereas the number of atomic propositions relevant in each component of the master automaton is constant, which we utilize and thus improve performance on this family both in terms of size and time.

ltl2ldba This translation is based on breakpoints for subformulae of the form $\mathbf{G}\psi$. We provide a heuristic that avoids breakpoints when ψ is a safety or co-safety subformula, see Table 3.

Besides, we add an option to generate a non-deterministic initial component for the LDBA instead of a deterministic one. Although the LDBA is then no more suitable for quantitative probabilistic model checking, it still is for qualitative model checking. At the same time, it can be much smaller, see Table 4 which shows a significant improvement on the particular formula.

ltl2dpa Both modes inherit the improvements of the respective **ltl2ldba** and **ltl2dgra** translations. Further, since complementing DPA is trivial, we can run in parallel both the translation of the input formula and of its negation, returning the smaller of the two results. Finally, we introduce several heuristics to optimize the treatment of safety subformulae of the input formula.

Table 4. Effect of non-determinism of the initial component for **ltl2ldba** on formulae $f(i) = \mathbf{F}(a \wedge \mathbf{X}^i\mathbf{G}b)$, displaying #states (#Büchi conditions)

	$f(1)$	$f(2)$	$f(3)$	$f(4)$	$f(5)$	$f(6)$
[SEJK16]	4 (1)	6 (1)	10 (1)	18 (1)	34 (1)	66 (1)
ltl2ldba	2 (1)	3 (1)	4 (1)	5 (1)	6 (1)	7 (1)

Table 5. Comparison of the average performance with the previous version of Rabinizer. The statistics are taken over a set of 200 standard formulae [KMS18] used, e.g., in [BKS13,EKS16], run in a batch mode for both tools to eliminate the effect of the JVM start-up overhead.

Tool	Avg # states	Avg # acc. sets	Avg runtime
Rabinizer 3.1 [EKS16]	6.3	6.7	0.23
ltl2dgra	6.2	4.4	0.12

dra2dpa The index appearance record of [KMWW17] keeps track of a permutation (ordering) of Rabin pairs. To do so, all ties between pairs have to be resolved. In our implementation, we keep a pre-order instead, where irrelevant ties are not resolved. Consequently, it cannot happen that an irrelevant tie is resolved in two different ways like in [KMWW17], thus effectively merging such states.

Implementation The main performance bottleneck of the older implementations is that explicit data structures for the transition system are not efficient for larger alphabets. To this end, Rabinizer 3.1 provided symbolic (BDD) representation of states and edge labels. On the top, Rabinizer 4 represents the transition function symbolically, too.

Besides, there are further engineering improvements on issues such as storing the acceptance condition only as a local edge labelling, caching, data-structure overheads, SCC-based divide-and-conquer constructions, or the introduction of parallelization for batch inputs.

Average performance evaluation We have already illustrated the improvements on several hand-crafted families of formulae. In Tables 1 and 2 we have even seen the respective running-time speed-ups. As the basis for the overall evaluation of the improvements, we use some established datasets from literature, see [KMS18], altogether two hundred formulae. The results in Table 5 indicate that the performance improved also on average among the more realistic formulae.

4 Conclusion

We have presented Rabinizer 4, a tool set to translate LTL to various deterministic automata and to use them in probabilistic model checking and in synthesis. The tool set extends the previous functionality of Rabinizer, improves on previous translations, and also gives the very first implementations of frequency LTL translation as well as model checking. Finally, the tool set is also more user-friendly due to richer input syntax, its connection to PRISM and PG Solver, and the on-line version with direct visualization, which can be found at `http://rabinizer.in.tum.de`.

References

[BBD+15] Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The hanoi omega-automata format. In *CAV, Part I*, pages 479–486, 2015.

[BBDL+13] Tomáš Babiak, Thomas Badie, Alexandre Duret-Lutz, Mojmír Křetínský, and Jan Strejček. Compositional approach to suspension and other improvements to LTL translation. In *SPIN*, pages 81–98, 2013.

[BDK+17] František Blahoudek, Alexandre Duret-Lutz, Mikuláš Klokočka, Mojmír Křetínský, and Jan Strejček. Seminator: A tool for semi-determinization of omega-automata. In *LPAR*, pages 356–367, 2017.

[BDL12] Benedikt Bollig, Normann Decker, and Martin Leucker. Frequency linear-time temporal logic. In *TASE*, pages 85–92, 2012.

[BKŘS12] Tomáš Babiak, Mojmír Křetínský, Vojtěch Řehák, and Jan Strejček. LTL to Büchi automata translation: Fast and more deterministic. In *TACAS*, pages 95–109, 2012.

[BKS13] Frantisek Blahoudek, Mojmír Křetínský, and Jan Strejček. Comparison of LTL to deterministic Rabin automata translators. In *LPAR*, volume 8312 of *LNCS*, pages 164–172, 2013.

[BMM14] Patricia Bouyer, Nicolas Markey, and Raj Mohan Matteplackel. Averaging in LTL. In *CONCUR*, pages 266–280, 2014.

[CGK13] Krishnendu Chatterjee, Andreas Gaiser, and Jan Křetínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In *CAV*, pages 559–575, 2013.

[CKK15] Krishnendu Chatterjee, Zuzana Komárková, and Jan Křetínský. Unifying two views on multiple mean-payoff objectives in markov decision processes. In *LICS*, pages 244–256, 2015.

[CY88] Costas Courcoubetis and Mihalis Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *FOCS*, pages 338–345, 1988.

[DLLF+16] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *ATVA*, pages 122–129, October 2016.

[EH00] Kousha Etessami and Gerard J. Holzmann. Optimizing Büchi automata. In *CONCUR*, pages 153–167, 2000.

[EK14] Javier Esparza and Jan Křetínský. From LTL to deterministic automata: A Safraless compositional approach. In *CAV*, pages 192–208, 2014.

[EKRS17] Javier Esparza, Jan Křetínský, Jean-Francois Raskin, and Salomon Sickert. From LTL and limit-deterministic Büchi automata to deterministic parity automata. In *TACAS*, pages 426–442, 2017.

[EKS16] Javier Esparza, Jan Kretínský, and Salomon Sickert. From LTL to deterministic automata - A safraless compositional approach. *Formal Methods in System Design*, 49(3):219–271, 2016.

[FKK15] Vojtěch Forejt, Jan Krčál, and Jan Křetínský. Controller synthesis for MDPs and frequency LTL\GU. In *LPAR*, pages 162–177, 2015.

[FL09] Oliver Friedmann and Martin Lange. Solving parity games in practice. In *ATVA*, pages 182–196, 2009.

[FL15] Dana Fisman and Yoad Lustig. A modular approach for büchi determinization. In *CONCUR*, pages 368–382, 2015.

[GKE12] Andreas Gaiser, Jan Křetínský, and Javier Esparza. Rabinizer: Small deterministic automata for LTL(F,G). In *ATVA*, pages 72–76, 2012.

[GL02] Dimitra Giannakopoulou and Flavio Lerda. From states to transitions: Improving translation of LTL formulae to Büchi automata. In *FORTE*, pages 308–326, 2002.

[GO01] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In *CAV*, pages 53–65, 2001. Tool accessible at `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/`.

[HLS+15] Ernst Moritz Hahn, Guangyuan Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. Lazy probabilistic model checking without determinisation. In *CONCUR*, volume 42 of *LIPIcs*, pages 354–367, 2015.

[JBB+17] Swen Jacobs, Nicolas Basset, Roderick Bloem, Romain Brenguier, Maximilien Colange, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Thibaud Michaud, Guillermo A. Pérez, Jean-François Raskin, Ocan Sankur, and Leander Tentrup. The 4th reactive synthesis competition (SYNTCOMP 2017): Benchmarks, participants & results. *CoRR*, abs/1711.11439, 2017.

[KE12] Jan Křetínský and Javier Esparza. Deterministic automata for the (F,G)-fragment of LTL. In *CAV*, volume 7358 of *LNCS*, pages 7–22, 2012.

[KK14] Zuzana Komárková and Jan Křetínský. Rabinizer 3: Safraless translation of LTL to small deterministic automata. In *ATVA*, volume 8837 of *LNCS*, pages 235–241, 2014.

[Kle] Joachim Klein. ltl2dstar - LTL to deterministic Streett and Rabin automata. `http://www.ltl2dstar.de/`.

[KLG13] Jan Křetínský and Ruslán Ledesma-Garza. Rabinizer 2: Small deterministic automata for LTL\GU. In *ATVA*, pages 446–450, 2013.

[KMS18] Jan Křetínský, Tobias Meggendorfer, and Salomon Sickert. LTL Store: Repository of LTL formulae from literature and case studies. *CoRR*, abs/1804.xxxx, 2018.

[KMWW17] Jan Křetínský, Tobias Meggendorfer, Clara Waldmann, and Maximilian Weininger. Index appearance record for transforming rabin automata into parity automata. In *TACAS*, pages 443–460, 2017.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

[KV15] Dileep Kini and Mahesh Viswanathan. Limit deterministic and probabilistic automata for LTL \ GU. In *TACAS*, pages 628–642, 2015.

[KV17] Dileep Kini and Mahesh Viswanathan. Optimal translation of LTL to limit deterministic automata. In *TACAS 2017*, 2017. To appear.

[ML16] Philipp J. Meyer and Michael Luttenberger. Solving mean-payoff games on the GPU. In *ATVA*, pages 262–267, 2016.

[MS17] David Müller and Salomon Sickert. LTL to deterministic Emerson-Lei automata. In *GandALF*, pages 180–194, 2017.

[Pit06] Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264, 2006.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[PP06] Nir Piterman and Amir Pnueli. Faster solutions of Rabin and Streett games. In *LICS*, pages 275–284, 2006.

[Saf88] Shmuel Safra. On the complexity of omega-automata. In *FOCS*, pages 319–327, 1988.

[SB00] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In *CAV*, pages 248–263, 2000.

[Sch09] Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *FoSSaCS*, volume 5504 of *LNCS*, pages 167–181, 2009.

[SEJK16] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Kretínský. Limit-deterministic büchi automata for linear temporal logic. In *CAV*, pages 312–332, 2016.

[SK16] Salomon Sickert and Jan Kretínský. Mochiba: Probabilistic LTL model checking using limit-deterministic büchi automata. In *ATVA*, pages 130–137, 2016.

[TD14] Cong Tian and Zhenhua Duan. Buchi determinization made tighter. Technical Report abs/1404.1436, arXiv.org, 2014.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344, 1986.