Comparison of LTL to Deterministic Rabin Automata Translators*

František Blahoudek, Mojmír Křetínský, and Jan Strejček

Faculty of Informatics, Masaryk University, Brno, Czech Republic
{xblahoud, kretinsky, strejcek}@fi.muni.cz

Abstract. Increasing interest in control synthesis and probabilistic model checking caused recent development of LTL to deterministic ω-automata translation. The standard approach represented by `ltl2dstar` tool employs Safra's construction to determinize a Büchi automaton produced by some LTL to Büchi automata translator. Since 2012, three new LTL to deterministic Rabin automata translators appeared, namely Rabinizer, LTL3DRA, and Rabinizer 2. They all avoid Safra's construction and work on LTL fragments only. We compare performance and automata produced by the mentioned tools, where `ltl2dstar` is combined with several LTL to Büchi automata translators: besides traditionally used LTL2BA, we also consider LTL->NBA, LTL3BA, and Spot.

1 Introduction

Linear temporal logic (LTL) has proved to be an appropriate formalism for specification of systems behavior with major applications in the area of model checking. Methods for LTL model checking of probabilistic systems [29, 5, 3] and for LTL synthesis [4, 24, 19] mostly need to construct, for any given LTL formula, a *deterministic* ω-automaton. As *deterministic Büchi automata (DBA)* cannot express all the properties expressible in LTL, one has to choose deterministic ω-automata with a more complex acceptance condition. The most common choice is the Rabin acceptance.

There are basically two approaches to translation of LTL to deterministic ω-automata. A traditional one translates LTL to *nondeterministic Büchi automata (NBA)* first and then it employs Safra's construction [26] (or some of its variants or alternatives like [23, 27]) to obtain a deterministic automaton. This approach is represented by the tool `ltl2dstar` [14] which uses an improved Safra's construction [16, 17]. As every LTL formula can be translated into an NBA and Safra's construction can transform any NBA to a *deterministic Rabin automaton (DRA)*, `ltl2dstar` works for the whole LTL. However, the resulting automata are sometimes unnecessarily big.

Since 2012, several translations avoiding Safra's construction have been introduced. The first one is presented in [18] and subsequently implemented in

* Authors are supported by The Czech Science Foundation, grant no. P202/10/1469.

the tool Rabinizer [10]. The algorithm builds a *generalized deterministic Rabin automaton (GDRA)* directly from a formula. A DRA is then produced by a degeneralization procedure. Rabinizer often produces smaller automata than ltl2dstar. The main disadvantage is that it works for LTL(F, G) only, i.e. the LTL fragment containing *eventually* (F) and *always* (G) as the only temporal operators. This method has been extended to a semantically larger fragment and reimplemented in the experimental tool Rabinizer 2 [21]. In [1] we present a Safraless translation working with another LTL fragment subsuming LTL(F, G). Our translator LTL3DRA transforms a given formula into a very weak alternating automaton (in the same way as LTL2BA [11]) and then into a *transition-based generalized deterministic Rabin automaton (TGDRA)*. The construction of generalized Rabin pairs of TGDRA is inspired by [18]. A DRA is finally obtained by a degeneralization procedure.

Here we provide a comparison of performance of the LTL to DRA translators ltl2dstar, Rabinizer, Rabinizer 2, and LTL3DRA. The tool ltl2dstar is designed to use an external LTL to NBA translator. To our best knowledge, the last experimental comparison of performance of ltl2dstar with different LTL to NBA translators has been done in 2005 [15]. The comparison shows that with respect to automata sizes, LTL2BA and LTL->NBA [9] “have the lead and were the only programs without failures to calculate the DRA.” Since 2005, significant progress has been made in LTL to NBA translation (it can already be seen in the comparison of LTL to NBA translators [25] published in 2007). Hence, we run ltl2dstar with LTL2BA, LTL->NBA, and contemporary translators Spot [6, 7] and LTL3BA [2]. The experimental results obtained are briefly interpreted.

2 Compared Tools

Here we describe settings and restrictions of the considered translators.

- ltl2dstar [14] v0.5.1, http://www.ltl2dstar.de/
  We keep the default setting (all optimizations enabled). We use only the option --ltl2nba="<intf>:<tool>[@<params>]" to specify an external <tool> for LTL to NBA translation (<intf> specifies if ltl2dstar communicates with the <tool> via the interface of *lbtt* [28] or *Spin* [13], and <params> are parameters the <tool> is called with). We use four LTL to NBA translators:
  - **LTL->NBA** [9], http://www.ti.informatik.uni-kiel.de/~fritz/
    We call it with --ltl2nba="lbtt:/pathtoLTL->NBA/script4lbtt.py".
  - **LTL2BA** [11] v1.1, http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/
    We call it with --ltl2nba="spin:/pathtoLTL2BA/ltl2ba".
  - **LTL3BA** [2] v1.0.2, http://sourceforge.net/projects/ltl3ba/
    By default, LTL3BA aims to produce small NBAs. With the option -M, it aims to produce potentially larger, but more deterministic automata. We have combined both modes with other optimizations provided by LTL3BA. We have selected two settings with the best results, namely --ltl2nba="spin:/pathtoLTL3BA/ltl3ba" referenced as LTL3BA and --ltl2nba="spin:/pathtoLTL3BA/ltl3ba@-M -S" referenced as LTL3BAd. Option -S enables strong fair simulation reduction.

- **Spot** [6, 7] v1.1.3, `http://spot.lip6.fr/wiki/`
  Again, Spot can be set to produce either small or more deterministic Büchi automata. We have combined `ltl2dstar` with both modes of Spot. The resulting Rabin automata produced with the first mode are usually identical to (and sometimes slightly bigger than) the automata produced with the latter mode. Computation times are also similar. To save some space, we include only the results for the "more deterministic" mode invoked by `--ltl2nba="spin:/pathtoSpot/ltl2tgba@-sD"`.

– **Rabinizer** [10] v0.11, `http://crab.in.tum.de/rabinizer/`
  Recall that Rabinizer works for LTL(F, G) only.

– **Rabinizer 2** [21],
  `http://www.model.in.tum.de/~kretinsk/rabinizer2.html`
  Rabinizer 2 works with formulae of a fragment called LTL\GU which uses not only F and G but also *next* (X) and *until* (U) temporal operators. The fragment consists of formulae in the negation normal form (i.e. negations are only in front of atomic propositions) such that no U is in the scope of any G.

– **LTL3DRA** [1] v0.1, `http://sourceforge.net/projects/ltl3dra/`
  This tool works with formulae of a slightly less expressive fragment than LTL\GU. More precisely, there is one more restriction on the scope of any G: there are no U operators, and X can appear only in front of F or G, i.e. in subformulae of the form $\mathsf{XF}\varphi$ or $\mathsf{XG}\varphi$. We call this fragment LTL\GUX. The difference is not important for specification formulae of software and asynchronous systems as these usually contain no X operators, but it can play some role in specification formulae of hardware and synchronous systems.

Before we run the translators, we transform input formulae to the expected format (prefix notation for `ltl2dstar` and negation normal form for Rabinizer 2) using the tool `ltlfilt` [7]. Note that Rabinizer, Rabinizer 2, and LTL3DRA are called with default settings.

3 Experiments: Benchmarks and Results

All experiments were done on a server with 8 eight-core processors Intel® Xeon® X7560, 2.26GHz, 448 GiB RAM and a 64-bit version of GNU/Linux. All the translators are single-threaded. The timeout limit was set to 2 hours.

We run the tools on three benchmark sets: real specification formulae, parametric formulae, and random formulae. The benchmark sets can be downloaded from the web pages of LTL3DRA.

Real specification formulae We use specification formulae from two sources: BEEM [22] and Spec Patterns [8]. After removing duplicates (typically cases where an atomic proposition a is consistently replaced by its negation or by $a \vee b$), we have 67 formulae. These formulae are divided into three classes: 12 formulae of LTL(F, G), 19 formulae of LTL\GUX not included in LTL(F, G), and 36 formulae outside LTL\GUX. Note that all the considered formulae outside LTL\GUX are also outside LTL\GU.

Unlike standard model checking algorithms, applications requiring deterministic ω-automata usually need automata equivalent to specification formulae and not to their negations. Hence, we do not negate the formulae before translation.

Table 1 presents cummulative results of the considered tools on the three classes of specification formulae. Table 2 provides a cross-comparison of the tools on the same formulae classes.

Class	Measure	ltl2dstar					Rabinizer	Rabinizer 2	LTL3DRA
		LTL->NBA	LTL2BA	LTL3BA	LTL3BAd	Spot			
12 formulae of LTL(F, G)	states	55	49	47	45	52	45	59	**43**
	edges	186	171	158	**151**	167	187	287	161
	pairs	18	18	**17**	**17**	**17**	22	18	21
	minimal	3	7	7	8	3	**10**	7	**10**
	time [s]	0.70	**0.12**	0.14	0.13	0.72	3.08	3.05	**0.12**
	mem max	22.53	**8.02**	18.66	18.69	91.06	240.75	465.09	19.02
	mem avg	19.66	**7.13**	18.57	18.61	86.92	160.03	173.53	18.90
19 more of LTL\GUX	states	180	191	184	167	**132**	—	160	137
	edges	614	699	671	563	**390**	—	827	546
	pairs	43	44	44	44	32	—	**28**	46
	minimal	2	2	2	3	6	—	**11**	**11**
	time [s]	2.83	0.24	0.32	0.30	2.11	—	5.98	**0.19**
	mem max	33.81	**8.72**	18.80	18.83	92.94	—	1 013.89	19.50
	mem avg	22.29	**7.44**	18.67	18.72	87.95	—	256.50	19.13
36 more of LTL	states	34 985	135 250	33 927	2 768	**386**	—	—	—
	edges	359 494	1 726 573	416 794	31 287	**1936**	—	—	—
	pairs	100	114	97	83	**49**	—	—	—
	minimal	9	8	9	13	**34**	—	—	—
	time [s]	26.46	102.15	16.86	**1.02**	1.64	—	—	—
	mem max	463.95	1 406.86	345.52	**24.41**	93.69	—	—	—
	mem avg	35.34	65.53	27.77	**18.90**	89.29	—	—	—

Table 1. For each class of considered real formulae and for each tool, the table shows cummulative numbers of *states*, *edges*, and accepting *pairs* of produced automata. Further, we show the number of *minimal* automata produced by the tool (minimal means that no other considered tool produced an automaton with less states for the same formula). We also provide cummulative computation *time* (in seconds) and maximal and average memory peaks (*mem max* and *mem avg*, measured in MiB) needed for the construction of one automaton. The best results are emphasized.

#	Tool		12 formulae of LTL(F, G)									19 more of LTL\GUX								36 more of LTL					
			1	2	3	4	5	6	7	8	V	1	2	3	4	5	7	8	V	1	2	3	4	5	V
1	ltl2dstar	LTL->NBA	—	0	0	0	0	1	3	1	5	—	1	1	2	0	4	3	11	—	13	9	3	0	25
2		LTL2BA	6	—	0	0	5	1	5	1	18	4	—	0	1	0	4	3	12	12	—	0	2	0	14
3		LTL3BA	6	1	—	0	5	1	5	1	19	4	1	—	1	0	4	3	13	14	14	—	4	0	32
4		LTL3BAd	6	1	1	—	6	1	5	1	21	4	2	2	—	0	5	4	17	22	17	13	—	2	54
5		Spot	1	1	0	0	—	1	4	1	8	12	9	9	8	—	7	6	51	27	28	27	23	—	**105**
6	Rabinizer		8	4	4	3	8	—	5	1	33	—	—	—	—	—	—	—	—	—	—	—	—	—	—
7	Rabinizer 2		6	3	3	3	6	0	—	1	22	15	15	15	14	10	—	4	**73**	—	—	—	—	—	—
8	LTL3DRA		9	4	4	3	9	2	5	—	**36**	14	12	12	11	9	8	—	66	—	—	—	—	—	—

Table 2. Cross-comparison of considered tools on the three classes of real specification formulae. The number in row indexed by r and column c represents in how many cases the tool r produced a smaller automaton (in the number of states) than the tool c. The column V shows the sum of these "victories".

Parametric formulae We consider 8 parametric formulae of [12] and formulae $\theta(n)$ of [11] and $F(n)$ of [18]:

$$E(n) = \bigwedge_{i=1}^{n} \mathsf{F} p_i \qquad C_1(n) = \bigvee_{i=1}^{n} \mathsf{GF} p_i$$
$$U(n) = (\ldots((p_1 \,\mathsf{U}\, p_2) \,\mathsf{U}\, p_3) \,\mathsf{U}\, \ldots) \,\mathsf{U}\, p_n \qquad C_2(n) = \bigwedge_{i=1}^{n} \mathsf{GF} p_i$$
$$R(n) = \bigwedge_{i=1}^{n} (\mathsf{GF} p_i \vee \mathsf{FG} p_{i+1}) \qquad Q(n) = \bigwedge_{i=1}^{n} (\mathsf{F} p_i \vee \mathsf{G} p_{i+1})$$
$$U_2(n) = p_1 \,\mathsf{U}\, (p_2 \,\mathsf{U}\, (\ldots (p_{n-1} \,\mathsf{U}\, p_n) \ldots)) \qquad S(n) = \bigvee_{i=1}^{n} \mathsf{G} p_i$$

$$\theta(n) = \neg((\bigwedge_{i=1}^{n} \mathsf{GF} p_i) \rightarrow \mathsf{G}(q \rightarrow \mathsf{F} r)) \qquad F(n) = \bigwedge_{i=1}^{n} (\mathsf{GF} p_i \rightarrow \mathsf{GF} q_i)$$

The results are shown in Table 3. Note that $U(n)$ and $U_2(n)$ are not in the input fragment of Rabinizer. All the other formulae are from LTL(F, G).

Formula	size	`ltl2dstar`					Rabinizer	Rabinizer 2	LTL3DRA
	max	LTL->NBA	LTL2BA	LTL3BA	LTL3BAd	Spot			
$E(n)$	$n = 9$	**512**	**512**	**512**	**512**	**512**	**512**	**512**	**512**
	max n	9	11	11	11	**12**	10	9	10
$U(n)$	$n = 5$	**17**	**17**	**17**	**17**	**17**	—	**17**	24
	max n	10	5	6	10	**12**	—	9	9
$R(n)$	$n = 3$	375 631	290 046	483 789	2 347	15 980	52	97	**36**
	max n	3	3	3	4	3	4	3	**6**
$U_2(n)$	$n = 14$	**15**	**15**	**15**	**15**	**15**	—	**15**	**15**
	max n	15	15	15	15	15	—	**19**	14
$C_1(n)$	$n = 7$	129	**2**	**2**	**2**	3	128	128	**2**
	max n	11	23	23	23	22	8	7	**24**
$C_2(n)$	$n = 6$	18	17	17	11	13	**7**	384	**7**
	max n	8	11	**17**	**17**	16	8	6	15
$Q(n)$	$n = 7$	1 331	1 140	1 140	1 140	736	**578**	**578**	2 790
	max n	7	8	8	8	**9**	8	7	7
$S(n)$	$n = 9$	513	513	513	513	513	**512**	**512**	**512**
	max n	**14**	**14**	**14**	**14**	11	9	9	13
$\theta(n)$	$n = 5$	21	20	15	5 444	5 444	11	480	**7**
	max n	7	10	**19**	6	6	7	5	14
$F(n)$	$n = 2$	13 181	11 324	5 650	302	4 307	20	32	**18**
	max n	2	2	2	2	2	3	2	**4**

Table 3. For each parametric formula and each tool, the table provides the *size* (number of states) of the automaton for the highest n such that all the considered tools finish the computation within the limit (upper row), and the *max*imal n for which the tool finishes the computation within the limit (lower row). The best values are emphasized.

Random formulae We use LTL formulae generator `randltl` [7] to get some more formulae of length 15–30 from various fragments. More precisely, we generate 100 formulae from the LTL(F, G) fragment, 100 general formulae with higher occurence of F and G operators, and 100 formulae with uniformly distributed operators. These three sets are generated by the respective commands:

- `randltl -n 100 --tree-size=15..30 --ltl-priorities="ap=1,X=0,\`
  `implies=0,false=0,true=0,R=0,equiv=0,U=0,W=0,M=0,xor=0" a b c d`
- `randltl -n 100 --tree-size=15..30 --ltl-priorities="ap=1,F=2,\`
  `G=2,false=0,true=0,X=1,R=1,U=1,W=0,M=0,xor=0" a b c d`

– `randltl -n 100 --tree-size=15..30 --ltl-priorities="ap=1,\`
  `false=0,true=0,W=0,M=0,xor=0" a b c d`

We removed 10 formulae, out of the 300 generated ones, that were elementary equivalent to *true* or *false*. The remaining formulae are divided into four classes corresponding to the input LTL fragments of the considered tools: we have 97 formulae of LTL(F, G), 29 formulae of LTL\GUX not included in LTL(F, G), 1 formula of LTL\GU not included in LTL\GUX, and 163 formulae not in LTL\GU. Unfortunately, `ltl2dstar` combined with LTL->NBA produces an error message for one formula of LTL\GUX and two formulae outside LTL\GU. These formulae were removed from the set. Further, there are 19 formulae (none of them in LTL\GU), for which at least one tool does not finish before timeout. These formulae are not included in the cummulative results to make them comparable, but we show the number of timeouts in a separate line. To sum up, Table 4 presents cummulative results for 97 formulae of LTL(F, G), 28 formulae of LTL\GUX not included in LTL(F, G), and 142 formulae outside LTL\GU (plus the numbers of timeouts for another 19 formulae outside LTL\GU). We do not show the results on the single formula of LTL\GU not included in LTL\GUX due to their low statistical significance.

Table 5 contains a cross-comparison of the tools on the same formulae sets. In this case, the formulae previously removed because of a timeout or a tool failure are included.

Class	Measure	`ltl2dstar`					Rabinizer	Rabin. 2	LTL3DRA
		LTL->NBA	LTL2BA	LTL3BA	LTL3BAd	Spot			
97 formulae of LTL(F, G)	states	107 620	19 470	9 914	6 008	13 940	**511**	741	618
	edges	949 094	165 856	76 827	48 440	137 977	**2222**	4 987	2 666
	pairs	217	204	196	190	164	198	**149**	198
	minimal	18	36	37	44	41	**54**	26	44
	time [s]	743.66	13.47	10.15	3.42	18.09	48.81	79.92	**1.21**
	mem max	6 561.89	151.16	99.75	24.86	94.03	406.66	6 712.00	**22.89**
	mem avg	95.72	**8.90**	19.51	18.77	89.27	205.10	632.62	19.23
28 more of LTL\GUX	states	1 183	6 670	6 375	1 509	633	—	**451**	512
	edges	6 227	39 987	38 591	8 057	3 002	—	**2422**	2 810
	pairs	66	68	69	54	**48**	—	71	70
	minimal	9	14	13	15	17	—	11	**18**
	time [s]	15.86	1.14	1.49	0.76	5.01	—	40.34	**0.50**
	mem max	107.75	45.83	41.53	**19.58**	94.17	—	33 224.44	34.59
	mem avg	39.80	**9.23**	19.63	18.87	89.72	—	1 761.70	20.07
142+19 more of LTL	states	173 156	640 971	157 869	143 436	**11780**	—	—	—
	edges	1 513 621	5 127 962	1 103 410	1 031 393	**85476**	—	—	—
	pairs	523	625	499	438	**354**	—	—	—
	minimal	54	41	57	72	**126**	—	—	—
	time [s]	421.79	384.54	76.33	70.38	**16.80**	—	—	—
	mem max	1 461.08	6 019.14	1 751.94	2 357.64	**99.50**	—	—	—
	mem avg	92.59	96.75	37.61	**35.45**	91.13	—	—	—
	timeouts	8	17	6	2	**1**	—	—	—

Table 4. The cummulative results on random formulae. Semantics of the table is the same as for Table 1. Moreover, the last line shows the number of *timeouts* of the tools on additional 19 formulae outside LTL\GU.

#		Tool	97 formulae of LTL(F, G)									29 more of LTL\GUX								163 more of LTL					
			1	2	3	4	5	6	7	8	*V*	1	2	3	4	5	7	8	*V*	1	2	3	4	5	*V*
1	`ltl2dstar`	LTL->NBA	—	13	10	6	2	10	35	17	93	—	4	5	4	1	6	6	26	—	79	43	38	16	176
2	`ltl2dstar`	LTL2BA	44	—	5	4	9	12	41	22	137	14	—	3	2	0	12	6	37	38	—	13	22	7	80
3	`ltl2dstar`	LTL3BA	44	17	—	5	11	13	43	23	156	14	3	—	1	0	11	5	34	68	80	—	30	16	194
4	`ltl2dstar`	LTL3BAd	48	24	18	—	15	15	45	28	193	15	6	6	—	2	14	8	51	87	97	73	—	24	281
5	`ltl2dstar`	Spot	52	31	26	16	—	19	46	32	222	18	8	9	7	—	15	8	65	106	115	99	74	—	**394**
6		Rabinizer	62	44	43	36	35	—	57	37	**314**	—	—	—	—	—	—	—	—	—	—	—	—	—	—
7		Rabinizer 2	42	23	19	20	19	2	—	26	151	17	10	10	6	7	—	5	55	—	—	—	—	—	—
8		LTL3DRA	58	43	40	33	35	13	47	—	269	17	11	12	10	9	14	—	**73**	—	—	—	—	—	—

Table 5. Cross-comparison of the considered tools on random formulae classes. The table has a similar semantics to Table 2: each number says in how many cases the tool in the corresponding row produces a *better* result than the tool in the corresponding column. An automaton is better than other if it has less states. Any automaton is better than timeout or a tool failure. Timeouts and failures are seen as equivalent results here.

4 Observations

For each pair of tools, there are some formulae in our benchmarks, for which one tool produces strictly smaller automata than the other (see Table 5). Hence, no tool is fully dominated by another.

All the results for LTL(F, G) fragment show that the Safraless tools (especially Rabinizer and LTL3DRA) usually perform better than `ltl2dstar` equipped with any of the considered LTL to NBA translators. The best results for formulae of LTL\GUX not included in LTL(F, G) are typically achived by `ltl2dstar` combined with Spot, and the Safraless tools Rabinizer 2 and LTL3DRA. For formulae outside LTL\GU, the current Safraless tools are not applicable. For these formulae, by far the best results are produced by `ltl2dstar` combined with Spot.

The results also provide information about particular tools or relations between them. For example, one can immediately see that Rabinizer outperforms Rabinizer 2 on LTL(F, G) formulae. This is explained by an experimental nature of the current version of Rabinizer 2. In particular, the tool misses some optimizations implemented in Rabinizer [20]. Further, one can observe that Rabinizer performs significantly better than the other tools on random formulae of LTL(F, G), while it is just comparable on real specification and parametric formulae of LTL(F, G). We assume that this is due to the fact that Rabinizer builds automata state-spaces according to semantics of LTL formulae rather than their syntax. Thus it does not distinguish between equivalent subformulae which more often appear in random formulae than in formulae written manually.

If we focus on usage of system resources, we observe that LTL3DRA is often the fastest tool. The results also show that `ltl2dstar` in combination with LTL2BA or LTL3BA has usually the lowest memory consumption.

During our experimentation we found out that `ltl2dstar` does not check whether an intermediate Büchi automaton is already deterministic or not: it runs Safra's construction in all cases. Running Safra's construction only on nondeterministic BA is profitable for two reasons:

1. Computation of Safra's construction is expensive.
2. Each deterministic BA can be directly converted into a DRA with one Rabin pair without any change in the state space, while Safra's construction typically produces a DRA larger than the intermediate deterministic BA.

For example, given the formula $\mathsf{G}(p_1 \rightarrow \mathsf{G}\neg p_2)$, both Spot and LTL3BAd produce a deterministic BA with two states (and a partial transition function). All considered LTL to DRA translators output DRA with four states (and total transition functions), Rabinizer 2 even yields a DRA with five states. Hence, the automaton produced by Spot or LTL3BAd is smaller even after the addition of one state to make its transition function total.

5 Conclusions

We conclude that the situation with LTL to DRA translation changed substantially since 2005. The former leading combinations of `ltl2dstar` with LTL->NBA or LTL2BA are now surpassed by Safraless tools (on relevant fragments) and `ltl2dstar` with Spot. However, there is still a space for further improvements.

Acknowledgements. We would like to thank Alexandre Duret-Lutz for valuable suggestions and comments on a draft of this paper.

References

1. T. Babiak, F. Blahoudek, M. Křetínský, and J. Strejček. Effective translation of LTL to deterministic Rabin automata: Beyond the (F,G)-fragment. In *ATVA'13*, volume 8172 of *LNCS*, pages 24–39. Springer, 2013.
2. T. Babiak, M. Křetínský, V. Řehák, and J. Strejček. LTL to Büchi automata translation: Fast and more deterministic. In *TACAS'12*, volume 7214 of *LNCS*, pages 95–109. Springer, 2012.
3. C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
4. A. Church. Logic, arithmetic, and automata. In *ICM'62*, pages 23–35. Institut Mittag-Leffler, 1962.
5. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.
6. A. Duret-Lutz. LTL translation improvements in Spot. In *VECoS'11*, Electronic Workshops in Computing. British Computer Society, 2011.
7. A. Duret-Lutz. Manipulating LTL formulas using Spot 1.0. In *ATVA'13*, volume 8172 of *LNCS*, pages 451–454. Springer, 2013.
8. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE'99*, pages 411–420. IEEE, 1999.
9. C. Fritz. Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata. In *CIAA'03*, volume 2759 of *LNCS*, pages 35–48. Springer, 2003.

10. A. Gaiser, J. Křetínský, and J. Esparza. Rabinizer: Small deterministic automata for LTL(F,G). In *ATVA'12*, volume 7561 of *LNCS*, pages 72–76, 2012.
11. P. Gastin and D. Oddoux. Fast LTL to Büchi Automata Translation. In *CAV'01*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001.
12. J. Geldenhuys and H. Hansen. Larger automata and less work for LTL model checking. In *SPIN'06*, volume 3925 of *LNCS*, pages 53–70. Springer, 2006.
13. G. Holzmann. *The Spin model checker: primer and reference manual.* Addison-Wesley Professional, first edition, 2003.
14. J. Klein. ltl2dstar – LTL to deterministic Streett and Rabin automata. http://www.ltl2dstar.de.
15. J. Klein. Linear time logic and deterministic omega-automata. Master's thesis, University of Bonn, 2005.
16. J. Klein and C. Baier. Experiments with deterministic ω-automata for formulas of linear temporal logic. *Theor. Comput. Sci.*, 363(2):182–195, 2006.
17. J. Klein and C. Baier. On-the-fly stuttering in the construction of deterministic ω-automata. In *CIAA'07*, volume 4783 of *LNCS*, pages 51–61. Springer, 2007.
18. J. Křetínský and J. Esparza. Deterministic automata for the (F, G)-fragment of LTL. In *CAV'12*, volume 7358 of *LNCS*, pages 7–22. Springer, 2012.
19. O. Kupferman. Recent challenges and ideas in temporal synthesis. In *SOFSEM'12*, volume 7147 of *LNCS*, pages 88–98. Springer, 2012.
20. J. Křetínský. Personal communication, 2013.
21. J. Křetínský and R. L. Garza. Rabinizer 2: Small deterministic automata for LTL\GU. In *ATVA'13*, volume 8172 of *LNCS*, pages 446–450. Springer, 2013.
22. R. Pelánek. Beem: Benchmarks for explicit model checkers. In *SPIN'07*, volume 4595 of *LNCS*, pages 263–267. Springer, 2007.
23. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007.
24. A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *ICALP'89*, volume 372 of *LNCS*, pages 652–671. Springer, 1989.
25. K. Y. Rozier and M. Y. Vardi. LTL Satisfiability Checking. In *SPIN'07*, volume 4595 of *LNCS*, pages 149–167. Springer, 2007.
26. S. Safra. On the complexity of omega-automata. In *FOCS'88*, pages 319–327. IEEE Computer Society, 1988.
27. S. Schewe. Tighter bounds for the determinisation of Büchi automata. In *FOSSACS'09*, volume 5504 of *LNCS*, pages 167–181. Springer, 2009.
28. H. Tauriainen and K. Heljanko. Testing LTL formula translation into Büchi automata. *International Journal on Software Tools for Technology Transfer (STTT)*, 4(1):57–70, 2002.
29. M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS'85*, pages 327–338. IEEE Computer Society, 1985.