EPiC Series in Computing

Volume 46, 2017, Pages 356–367

LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Seminator: A Tool for Semi-Determinization of Omega-Automata

František Blahoudek[1], Alexandre Duret-Lutz[2], Mikuláš Klokočka[1], Mojmír Křetínský[1], and Jan Strejček[1]

[1] Masaryk University, Brno, Czech Republic
{xblahoud, xklokock, kretinsky, strejcek}@fi.muni.cz
[2] LRDE, EPITA, Le Kremlin-Bictre, France
adl@lrde.epita.fr

Abstract

We present a tool that transforms nondeterministic ω-automata to semi-deterministic ω-automata. The tool Seminator accepts transition-based generalized Büchi automata (TGBA) as an input and produces automata with two kinds of semi-determinism. The implemented procedure performs degeneralization and semi-determinization simultaneously and employs several other optimizations. We experimentally evaluate Seminator in the context of LTL to semi-deterministic automata translation.

1 Introduction

Semi-deterministic automata (also known as *limit-deterministic* or *deterministic in the limit*) have been introduced more than 30 years ago [14]. They are considered mainly in the context of model checking of probabilistic systems, where nondeterministic automata are inapplicable and deterministic automata are often unnecessarily large. More precisely, semi-deterministic automata are known to be convenient for qualitative model checking of *Markov decision processes (MDPs)* [18, 5].

Semi-deterministic automata were neglected for a long time, as they do not bring any substantial theoretical advantage over deterministic automata. Indeed, an LTL formula can be translated into a doubly exponential semi-deterministic automaton as well as into a doubly exponential deterministic automaton. However, with the development of practically usable tools for analysis of probabilistic systems, semi-deterministic automata gained a new wave of interest. During the last two years, there appeared: a direct translation of an LTL fragment (LTL\GU) to semi-deterministic automata [12], an algorithm for quantitative model checking of MDPs based on semi-deterministic automata [10], a procedure for efficient complementation of semi-deterministic automata [4], a direct translation of full LTL to semi-deterministic automata of a specific form (called *cut-deterministic* automata later) [15] implemented in a tool ltl2ldba and presented together with a modification of a standard algorithm for quantitative model checking of MDPs employing these automata, and an implementation of this algorithm in a tool MoChiBA [16].

In practice, a specification property for a model checking procedure is typically given as an LTL formula. The formula can be translated to a semi-deterministic automaton directly, using the tool ltl2ldba [15]. Another possibility is to translate the formula into a nondeterministic automaton and then apply a semi-determinization procedure [5, 10]. While there exist optimized tools like Spot [7] or LTL3BA [3] for translation of LTL to nondeterministic automata, we did not know about any tool for semi-determinization of these automata when we submitted the paper. Seminator was introduced to fill this gap. The tool nba2ldba for semi-determinization of Büchi automata appeared recently. Experimental results presented in this paper show that Seminator performs better.

Seminator implements semi-determinization procedures similar to the one presented by Hahn et al. [10]. The procedures take a *transition-based generalized Büchi automaton (TGBA)* as an input. Note that the two LTL-to-nondeterministic automata translators mentioned above (Spot and LTL3BA) can produce TGBAs and these automata are usually smaller than equivalent *Büchi automata (BA)*. A TGBA can be degeneralized into a BA and then semi-determinized by the standard procedure [5], but doing both steps at once often results in a smaller automaton.

Seminator can produce two kinds of semi-deterministic automata. First, automata where all the states reachable from accepting states (or behind accepting transitions in the case of transition-based acceptance) are deterministic. These *semi-deterministic* automata are suitable for qualitative model checking of MDPs. Second, *cut-deterministic* automata where nondeterminism is even more restricted. Section 2 defines these two types of automata. The implemented semi-determinization and cut-determinization procedures are given in Section 3. Section 4 describes the tool and Section 5 presents an experimental comparison of Seminator and ltl2ldba.

2 Semi-Deterministic Automata

A *transition-based generalized Büchi automaton (TGBA)* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_I, \mathcal{F})$, where Q is a finite set of *states*, Σ is a finite *alphabet*, $\delta \subseteq Q \times \Sigma \times Q$ is a *transition relation*, $q_I \in Q$ is the initial state, and $\mathcal{F} = \{F_1, \ldots, F_n\}$ with $F_1, \ldots, F_n \subseteq \delta$ are sets of accepting transitions. A TGBA with a single set of accepting transitions is sometimes called *TBA*.

A run ρ of a TGBA $\mathcal{A}$ over a word $w = w_0 w_1 w_2 \ldots \in \Sigma^\omega$ is an infinite sequence of adjoining transitions $\rho = (q_0, w_0, q_1)(q_1, w_1, q_2) \ldots \in \delta^\omega$ where $q_0 = q_I$. A run ρ is *accepting* if, for each accepting set $F_i \in \mathcal{F}$, it contains infinitely many transitions from F_i. A word w is *accepted* if there is an accepting run over w. The *language* of a TGBA $\mathcal{A}$ is the set $L(\mathcal{A})$ of all words accepted by $\mathcal{A}$.

As illustrated by Figure 1, a TGBA $\mathcal{A} = (Q, \Sigma, \delta, q_I, \mathcal{F})$ is *semi-deterministic* if Q is a union of two disjoint sets Q_N and Q_D such that

- there is no transition from Q_D to Q_N, i.e., $\delta \cap (Q_D \times \Sigma \times Q_N) = \emptyset$,
- states in Q_D are deterministic, i.e., for each $p \in Q_D$ and $a \in \Sigma$ there exists at most one $q \in Q$ satisfying $(p, a, q) \in \delta$, and
- accepting transitions are between states of Q_D, i.e., $F_i \subseteq Q_D \times \Sigma \times Q_D$ holds for each $F_i \in \mathcal{F}$.

A TGBA is called *cut-deterministic* if it is semi-deterministic and moreover it holds that the states in Q_N are also deterministic when considering only the transitions leading to states of Q_N, i.e., for each $p \in Q_N$ and $a \in \Sigma$ there exists at most one $q \in Q_N$ satisfying $(p, a, q) \in \delta$. Intuitively, nondeterminism in a cut-deterministic automaton can be induced only by transitions

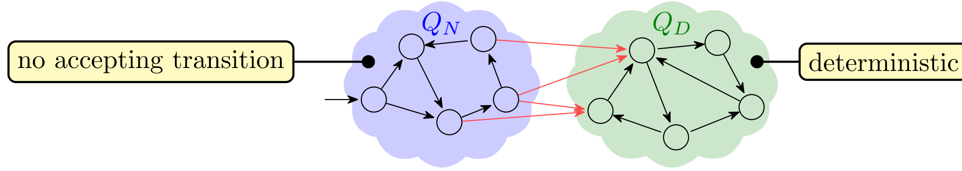

Figure 1: Structure of a semi-deterministic automaton. The green cloud is deterministic and contains all accepting transitions and states that are reachable from them. In a cut-deterministic automaton, the blue cloud is deterministic too.

from Q_N to Q_D. Note that the term *cut-determinism* is just introduced here (unused before) and we propose it to disambiguate the overloaded term *semi-determinism* [19, 10, 4]. It is inspired by the graph-theoretic notion of cut (the red transitions of Figure 1).

Finally, an automaton is called *deterministic* if all its states are deterministic. Clearly, every deterministic automaton is also cut-deterministic and every cut-deterministic automaton is also semi-deterministic. The opposite relations do not hold.

3 Semi-Determinization and Cut-Determinization

We first present a procedure that transforms a TGBA into a semi-deterministic TBA. Then we modify the procedure to produce a cut-deterministic TBA. Finally, we discuss correctness of the procedures.

3.1 Semi-Determinization

Let $\mathcal{A} = \big(Q, \Sigma, \delta, q_I, \mathcal{F}\big)$ be a TGBA where $\mathcal{F} = \{F_1, \ldots, F_n\}$ for some $n > 0$. Then an equivalent semi-deterministic TBA $\mathcal{B} = \big(Q', \Sigma, \delta', q_I, \{F'\}\big)$ can be constructed as follows.

- We set $Q' = Q \cup D$, where $D = 2^Q \times \{0, \ldots, n-1\} \times 2^Q$. States $(M, i, N) \in D$ form the deterministic part of automaton $\mathcal{B}$. The set M tracks runs of $\mathcal{A}$ using the standard subset construction. The *level* i has a similar meaning as in a standard degeneralization procedure: it records that each state in M can be reached by a run that passed through sets $F_1, \ldots, F_i$ since the last accepting transition of $\mathcal{B}$ was taken. Finally, the set $N \subseteq M$ tracks the runs that also passed the acceptance set F_{i+1}.

- The transition relation δ' contains three kinds of transitions:

  (1) All transitions of δ are also in δ'.

  (2) For every transition $(p, a, q) \in F_1$, we add a transition leading from p to the state in the deterministic part of $\mathcal{B}$ that corresponds to q and with the level after passing F_1. Formally, we add the transition $\big(p, a, (\{q\}, 1 \bmod n, \emptyset)\big) \in \delta'$.

  (3) We construct deterministic transitions between the states of D. For each $(M, i, N) \in D$ and $a \in \Sigma$, we compute the set $M' = \bigcup_{p \in M}\{q \mid (p, a, q) \in \delta\}$ of all successors of the states in M under a. If M' is not empty, the transition $\big((M, i, N), a, (M', i', N')\big)$ is added to δ', where i' and N' are constructed as follows. We first compute the set N'' of all successors of the states in N under a and we add there also all successors

of the states of M reached by a-transitions in F_{i+1}. Formally,

$$N'' = \bigcup_{p \in N} \{q \mid (p, a, q) \in \delta\} \cup \bigcup_{p \in M} \{q \mid (p, a, q) \in F_{i+1}\}.$$

If $N'' = M'$, then each tracked state can be reached by a run that already passed F_{i+1} and we move to the next level, i.e., $i' = (i + 1) \bmod n$ and $N' = \bigcup_{p \in M}\{q \mid (p, a, q) \in F_{i'+1}\}$. Otherwise, we set $N' = N''$ and $i' = i$.

- The set F' of accepting transitions consists of the transitions between states of D where $N'' = M'$ and $i' = 0$. Such a transition is taken if all tracked states can be reached by runs of $\mathcal{A}$ that passed all accepting sets $F_1, \ldots, F_n$.

3.2 Cut-Determinization

A cut-deterministic TBA $\mathcal{B}$ equivalent to a TGBA $\mathcal{A}$ can be obtained by the procedure given above supplemented with a subset construction to determinize the first part of the automaton. Formally, we build $\mathcal{B} = \big(Q', \Sigma, \delta', \{q_I\}, \{F'\}\big)$, where all symbols have the same meaning as before except the following.

- $Q' = 2^Q \cup D$ and D is defined as before.
- The construction of parts (1) and (2) of δ' is modified, while the part (3) remains unchanged.
  (1) For each $M \in 2^Q$ and $a \in \Sigma$, we compute the set $M' = \bigcup_{p \in M}\{q \mid (p, a, q) \in \delta\}$. If M' is not empty, we add the transition $(M, a, M') \in \delta'$.
  (2) We construct a transition entering D from a state M if there is a transition in F_1 leading from some state of M. Formally, for every transition $(p, a, q) \in F_1$ and for every M containing p, we add the transition $\big(M, a, (\{q\}, 1 \bmod n, \emptyset)\big) \in \delta'$.

3.3 Correctness

One can readily confirm that the automata constructed by the described procedures are indeed semi-deterministic and cut-deterministic, respectively.

The proof of language equivalence of an input automaton $\mathcal{A}$ and the constructed automaton $\mathcal{B}$ is more complex. While it should be relatively easy to see that every accepting run of $\mathcal{B}$ subsumes some accepting run of $\mathcal{A}$ and thus $L(\mathcal{B}) \subseteq L(\mathcal{A})$, the opposite language inclusion is non-trivial. Formal proofs of correctness would be very similar to these by Hahn et al. [10] as our semi-determinization procedures differ only in details.

4 Seminator

Seminator implements the algorithms described in Section 3 with some additional optimizations that are described in Section 4.1. The tool takes a TGBA on input and returns an equivalent semi-deterministic or cut-deterministic automaton as output. The tool is implemented in C++ using the Spot library [7] and is distributed under the *GNU GPL v3* license. It relies on *Hanoi Omega-Automata (HOA)* format [2] for reading and writing automata. The source code, installation, and usage instructions can be found on the tool's web page which is listed in Table 1.

There are two choices for the Seminator users regarding the type of automata on output.

1. Seminator outputs semi-deterministic automata by default. A cut-deterministic automaton can be requested by the option `--cd`.

2. TGBA are produced by default, transition-based Büchi automata can be requested by `--tba` and state-based Büchi automata by `--ba` if needed. In fact, a TGBA is returned only if Seminator does not perform the full semi-determinization (see Section 4.1 for more details).

We have two possible ways how to use our algorithms for transformation of a TGBA into an equivalent automaton of requested type. Besides running them directly on the TGBA, one can first transform the TGBA into an equivalent BA and run the algorithm on the BA. The latter approach is basically equivalent to the semi-determinization procedure described by Courcoubetis and Yannakakis [5]. This behavior can be requested by the option `--cy`.

The size of the produced automata can be often further improved by the automata reduction techniques that are implemented in Spot. They all preserve semi-determinism of TGBAs. However, the reverse simulation technique [1] does not preserve cut-determinism and thus it is not applied if cut-deterministic automaton is requested. The automata reduction techniques can be disabled by the option `-s0`.

4.1 Optimizations

Seminator applies few straightforward optimizations that can result in smaller automata on output. Clearly we do not need to modify the input automaton if it already complies with the requested type. In such cases, only simplifications offered by Spot are applied. Moreover, before testing the input automaton for semi- or cut-determinism we apply the following language-preserving modification of the automaton: *We remove from accepting sets $F_1, \ldots, F_n$ each transition that is not inside any accepting strongly connected component, where a component is accepting if it contains at least one transition of each accepting set F_i.* This modification itself can transform an automaton which is not semi-deterministic into a semi-deterministic one (the same holds for cut-determinism) and even if it is not the case, it can reduce the size of resulting automaton as smaller part of the original automaton has to be determinized.

The full cut-determinization can be avoided even in cases when a cut-deterministic automaton is requested and the input automaton is semi-deterministic but not cut-deterministic. In such case, we first compute a division of Q into Q_N and Q_D that complies with the definition of semi-determinism such that the deterministic part is as large as possible. There are no accepting transitions in the first part, thus we can apply the classical subset construction to states and transitions of Q_N. This will result in two separate deterministic components. Now we only have to add transitions from Q_N to Q_D to preserve the language properly. To be more precise, if there was a transition (q, a, p) from Q_N to Q_D in the original automaton, we add a transition (M, a, p) in the new automaton for each M that contains q. Note that semi-determinization and cut-determinization always produce a TBA. Seminator can produce a TGBA with two or more accepting sets only when these constructions are avoided due to the above optimizations.

An early version of Seminator sometimes produced a smaller automaton when executed with `--cy` option, i.e., when an input automaton is first degeneralized into a BA. This is because Seminator calls the highly optimized degeneralization procedure implemented in Spot [1], which is not the case when degeneralization is performed simultaneously with the semi-determinization or cut-determinization described in Section 3. We have modified Seminator to use the following three modes of dealing with degeneralization, compare the resulting automata sizes and return the smallest automaton out of the three.

Table 1: Tools used in the experimental evaluation.

tools	version	webpage
Seminator	1.1.0	https://github.com/mklokocka/seminator/
ltl2ldba, nba2ldba	1.0.0	https://www7.in.tum.de/~sickert/projects/owl/
ltl2tgba, autfilt	2.3.2	https://spot.lrde.epita.fr/

1. Convert directly the input TGBA.
2. Create an equivalent TBA and then perform the conversion.
3. Degeneralize into a BA and then perform the conversion.

5 Experimental Evaluation

In this section we evaluate Seminator and compare it to existing work. To our best knowledge, only two other tools can produce semi-deterministic or cut-deterministic automata and are thus relevant for comparison with Seminator. The tool nba2ldba converts BA into semi-deterministic BA and ltl2ldba translates LTL formulae directly to semi-deterministic or cut-deterministic TGBA [15]. Both tools are now distributed as parts of the Owl library (see Table 1).

5.1 Experimental Setup and Results

Because ltl2ldba needs an LTL formula on input, our evaluation starts with LTL formulae and translates them by Spot's `ltl2tgba -D` to automata expected on input of other tools. Option `-D` expresses a preference towards more deterministic output, but does not guarantee it.

We use two benchmark sets of LTL formulae. The first set consists of formulae collected from literature [8, 13, 9, 17, 11]. For each formula from the sources we added its negation into our set. We further simplified all the formulae by ltlfilt [6], removed duplicates and formulae equivalent to true or false. The resulting benchmark set contains 222 formulae. Figure 2 shows that it is very often the case that `ltl2tgba -D` produces a deterministic TGBA (●), or a non-deterministic TGBA that is already cut-deterministic (▲). Depending on its configuration, Seminator only has to perform some work on automata that are not cut-deterministic (■ and ◆) or on automata that are not semi-deterministic (◆).

Because there are few formulae on which Seminator actually has to work in the previous set, we use a second set of formulae generated randomly, but filtered so that each of the four types of `ltl2tgba -D` output (●, ▲, ■, ◆) has exactly 100 formulae. The files with all formulae used in this evaluation can be found in the GitHub repository of Seminator.

Table 3 compares the sizes (number of states) of semi-deterministic automata produced by Seminator and ltl2ldba, and nba2ldba in configurations given in Table 2. One trick used in Seminator is that it can perform degeneralization of the input TGBA simultaneously with semi-determinization or cut-determinization. To see the impact of this, we also include Seminator with the option `--cy` into our evaluation. This setting mimics the construction of Courcoubetis and Yannakakis [5] applied after degeneralization and is referenced by the acronym CY in the evaluation.

Another particularity of Seminator is that after applying semi-determinisation (or cut-determinisation) described in Section 3, it reduces the resulting automaton using the sim-

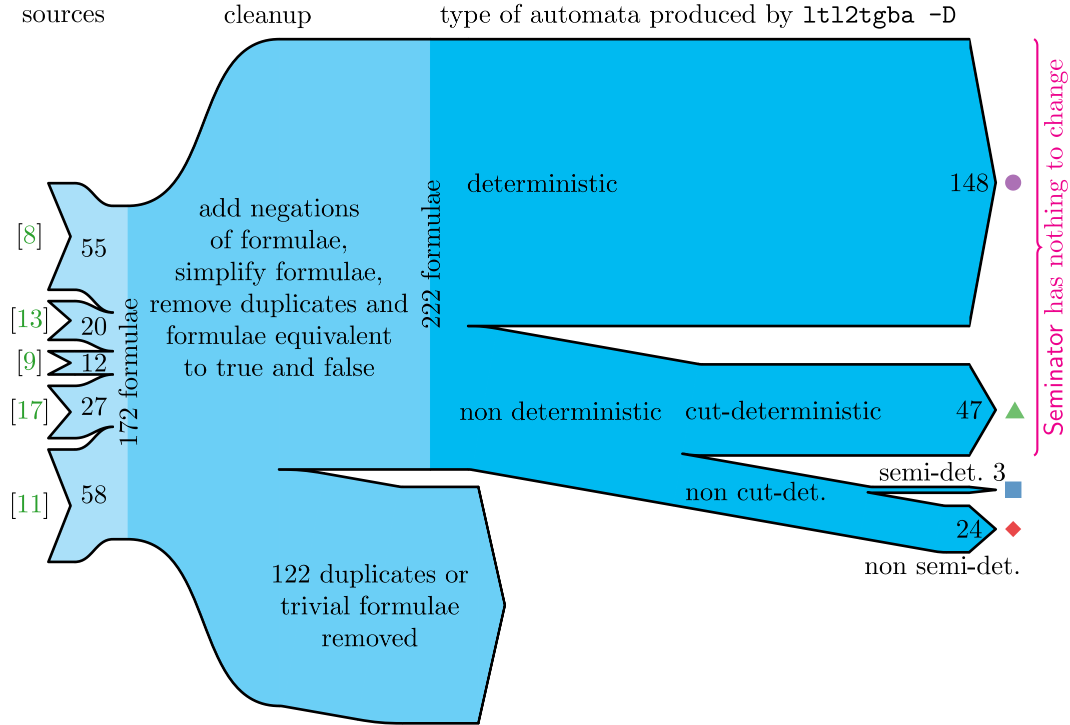

Figure 2: Preparation of the formulae from the literature, and classification according to the four types of automata produced by `ltl2tgba -D`.

Table 2: Tool configurations for generating a semi-deterministic automaton from formula φ.

approach	reductions	command line
Seminator	no	`ltl2tgba -D` φ `| seminator -s0`
	yes	`ltl2tgba -D` φ `| seminator`
CY	no	`ltl2tgba -D` φ `| seminator --cy -s0`
	yes	`ltl2tgba -D` φ `| seminator --cy`
ltl2ldba	no	`ltl2ldba -n` φ
	yes	`ltl2ldba -n` φ `| autfilt -D`
nba2ldba	no	`ltl2tgba -B -D` φ `| nba2ldba`
	yes	`ltl2tgba -B -D` φ `| nba2ldba | autfilt -D`

plification procedures of Spot. These reductions can have a strong effect on the size of the produced automata, therefore, to ease comparison with other tools, we evaluate each tool with and without these reductions. As shown in Table 2, the reductions can be disabled with option `-s0` in Seminator, and can be applied to other tools by passing their result through `autfilt`.

The evaluation ran on a desktop computer with Intel i7-3770 (3.40 GHz) processor and 8GB RAM. All toolchains finished the computation for each but one input formula within one minute. For one formula from literature, the reverse-simulation based reduction of semi-

Table 3: Evaluation of the tools producing semi-deterministic automata, on random LTL formulae and LTL formulae from literature classified according the type of automata produced by `ltl2tgba -D`. Each cell presents the cummulative size (number of states) of semi-deterministic automata produced by the corresponding tool without ('no') or with ('yes') reductions for the corresponding set of n formulae.

formulae			CY		ltl2ldba		nba2ldba		Seminator	
origin	type	n	no	yes	no	yes	no	yes	no	yes
random	● det	100	426	426	664	442	530	426	413	413
	▲ cd	100	510	510	715	535	816	510	467	467
	■ sd	100	720	720	1228	787	1085	720	704	704
	◆ nd	100	3408	1637	1666	873	3539	1788	3209	1500
literature	● det	148	596	596	1263	851	816	596	555	555
	▲ cd	47	211	211	834	344	363	211	197	197
	■ sd	3	13	13	49	17	17	13	13	13
	◆ nd	23	687	418	616	326	777	459	608	400
lit. (T/O)	◆ nd	1	148	—	49	49	164	—	115	—

Table 4: Tool configurations for generating cut-deterministic automata. (The `autfilt` invocation has extra options to disable reverse-simulation based reductions, since those do not preserve cut-determinism.)

approach	reductions	command line
Seminator	no	`ltl2tgba -D` φ `| seminator --cd -s0`
	yes	`ltl2tgba -D` φ `| seminator --cd`
CY	no	`ltl2tgba -D` φ `| seminator --cy --cd -s0`
	yes	`ltl2tgba -D` φ `| seminator --cy --cd`
ltl2ldba	no	`ltl2ldba` φ
	yes	`ltl2ldba` φ `| autfilt -D -xsimul=1,ba-simul=1`

deterministic automata produced by Seminator, CY, and nba2ldba did not finished within this timeout. Table 3 shows computed values for this formula separately in the last line.

Tables 4 and 5 show configurations and evaluation results for tools set to output cut-deterministic automata.

Further, Figure 3 provides comparison of Seminator and ltl2ldba on the level of individual semi-deterministic or cut-deterministic automata produced for considered formulae. Both tools run without reductions to expose the difference of core algorithms of the tools. Finally, Figure 4 compares semi-deterministic automata produced by Seminator the those produced by nba2ldba. Again, both tool run without reductions.

5.2 Observations

The presented results immediately lead to several observations.

1. Seminator produces nearly always the smallest semi-deterministic or cut-deterministic

Table 5: Evaluation of the tools producing cut-deterministic automata, on random LTL formulae and LTL formulae from literature classified according the type of automata produced by `ltl2tgba -D`. Each cell presents the cummulative size (number of states) of cut-deterministic automata produced by the corresponding tool without ('no') or with ('yes') reductions for the corresponding set of n formulae.

formulae			CY		ltl2ldba		Seminator	
origin	type	n	no	yes	no	yes	no	yes
random	● det	100	426	426	566	493	413	413
	▲ cd	100	510	510	730	650	467	467
	■ sd	100	750	728	1492	1277	734	712
	◆ nd	100	4342	1954	1394	1042	4032	1754
literature	● det	148	596	596	1033	806	555	555
	▲ cd	47	211	211	610	493	197	197
	■ sd	3	13	13	61	40	13	13
	◆ nd	24	1214	661	469	409	907	554

automaton if it gets on input a TGBA that is already semi-deterministic (which includes deterministic and cut-deterministic automata as well). Note that Seminator does not change such automata at all unless a cut-deterministic automaton is required and it gets a semi-deterministic automaton that is not cut-deterministic. In this case, Seminator just applies the subset construction on the nondeterministic part of the automaton. Hence, all these results reflect the efficiency of Spot's LTL to TGBA translation and not the efficiency of the Seminator's core algorithm.

2. When Seminator gets a TGBA that is not semi-deterministic, it usually produces a bigger cut-deterministic automaton than the one produced by ltl2ldba directly from the formula. When semi-deterministic automata are produced, the situation is similar, only the dominance of ltl2ldba is slightly smaller. Note that Seminator always produces a TBA in these cases, while ltl2ldba produces a TGBA.

3. Numbers in Tables 3 and 5 show that reductions can save many states of semi-deterministic and cut-deterministic automata produced by Seminator, ltl2ldba, or nba2ldba.

4. The obtained semi-deterministic automata are not dramatically smaller than the corresponding cut-deterministic automata.

5. Semi-deterministic automata produced by ltl2ldba can be bigger than cut-deterministic automata produced by the same tool. This is unexpected and it indicates a potential for further improvement of the tool.

The experimental evaluation brought two main outputs. First, if someone needs to translate an LTL formula to a small semi-deterministic automaton, it pays to try to translate it by Spot. If Spot produces a semi-deterministic automaton, it is very probably smaller than what ltl2ldba would produce. The same holds when cut-deterministic automaton is needed, but it may be necessary to run Seminator to cut-determinize the semi-deterministic automaton produced by Spot. Second, if someone needs to get a semi-deterministic automaton from a nondeterministic automaton rather than from an LTL formula, Seminator will probably deliver a smaller automaton than nba2ldba.

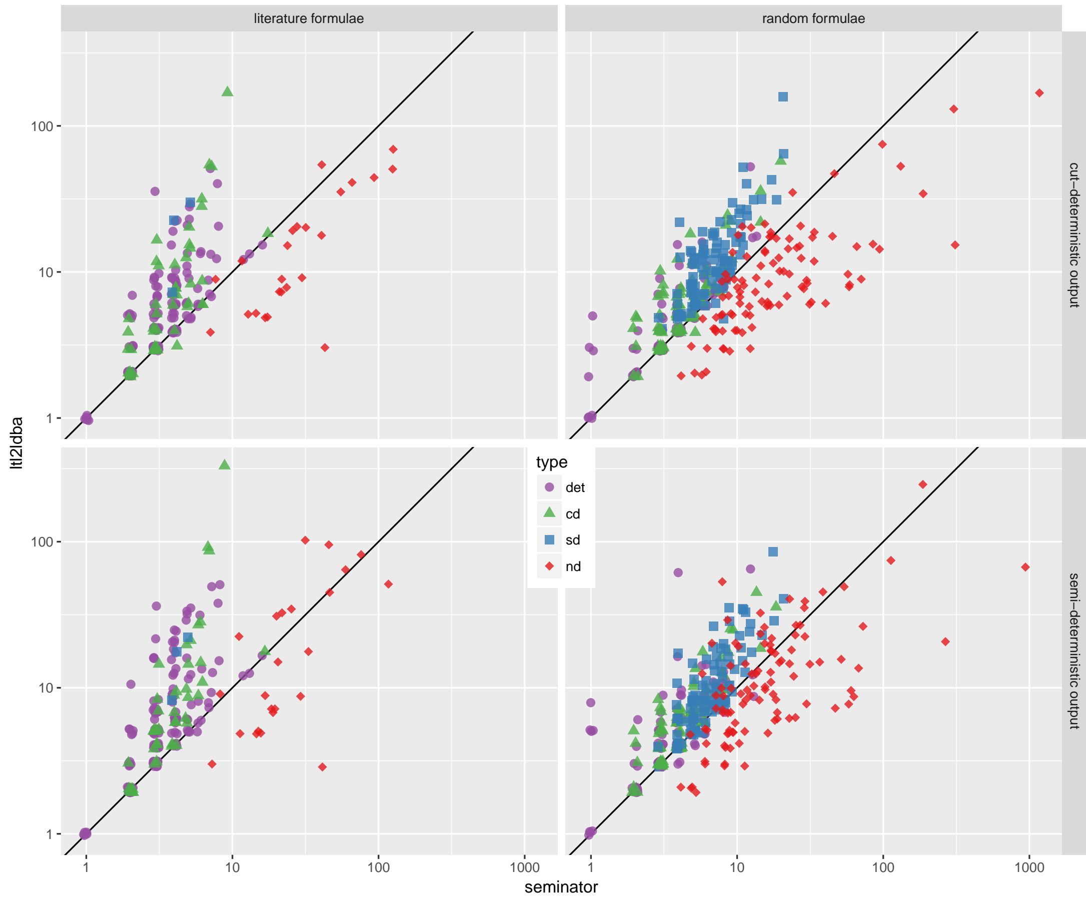

Figure 3: Comparison of the size of cut-deterministic automata produced by Seminator and ltl2ldba (both without reductions) on random formulae and on formulae from literature, and the analogous comparison of produced semi-deterministic automata. Scatter plots are colored according to the output type of `ltl2tgba -D`. Scales are logarithmic.

6 Conclusion

We introduced a tool called Seminator for semi-determinization of nondeterministic (transition-based generalized) Büchi automata. In combination with the LTL to automata translator in Spot, Seminator often produces smaller automata than the direct LTL to semi-deterministic automata translator ltl2ldba. However, most of these cases are due to the highly optimized translation of Spot.

When comparing to the other available automata semi-determinization tool nba2ldba, Seminator usually performs better. It also offers more flexibility for input automata and support of two kinds of semi-determinism.

We plan to further improve Seminator, in particular to adopt heuristics for better degeneralization of generalized Büchi acceptance.

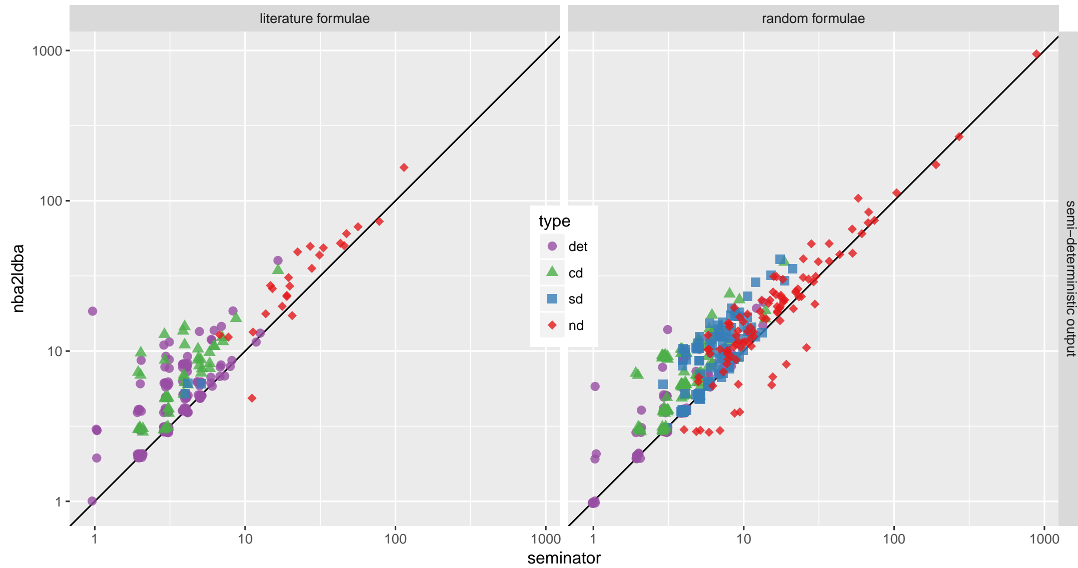

Figure 4: Comparison of the size of semi-deterministic automata produced by Seminator and nba2ldba (both without reductions) on random formulae and on formulae from literature. Scatter plots are colored according to the output type of `ltl2tgba -D`. Scales are logarithmic.

Acknowledgments The authors would like to thank Salomon Sickert for pointing out that not every cut-deterministic automaton is suitable for quantitative model checking of MDPs. F. Blahoudek, M. Klokočka, M. Křetínský, and J. Strejček have been supported by the Czech Science Foundation grant GBP202/12/G061.

References

[1] T. Babiak, T. Badie, A. Duret-Lutz, M. Křetínský, and J. Strejček. Compositional approach to suspension and other improvements to LTL translation. In *Model Checking Software - 20th International Symposium, SPIN 2013*, volume 7976 of *LNCS*, pages 81–98. Springer, 2013.

[2] T. Babiak, F. Blahoudek, A. Duret-Lutz, J. Klein, J. Křetínský, D. Müller, D. Parker, and J. Strejček. The Hanoi omega-automata format. In *Computer Aided Verification - 27th International Conference, CAV 2015*, volume 9206 of *LNCS*, pages 479–486. Springer, 2015.

[3] T. Babiak, M. Křetínský, V. Řehák, and J. Strejček. LTL to Büchi automata translation: Fast and more deterministic. In *Proc. of the 18th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214 of *LNCS*, pages 95–109. Springer, 2012.

[4] F. Blahoudek, M. Heizmann, S. Schewe, J. Strejček, and M. Tsai. Complementing semi-deterministic Büchi automata. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016*, volume 9636 of *LNCS*, pages 770–787. Springer, 2016.

[5] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *29th Annual Symposium on Foundations of Computer Science (FOCS'88)*, pages 338–345. IEEE Computer Society, 1988.

[6] A. Duret-Lutz. Manipulating LTL formulas using Spot 1.0. In *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13)*, volume 8172 of *Lecture Notes in Computer Science*, pages 442–445. Springer, 2013.

[7] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *LNCS*, pages 122–129. Springer, 2016.

[8] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In M. Ardis, editor, *Proceedings of the 2nd Workshop on Formal Methods in Software Practice (FMSP'98)*, pages 7–15, New York, Mar. 1998. ACM Press.

[9] K. Etessami and G. J. Holzmann. Optimizing Büchi automata. In C. Palamidessi, editor, *Proceedings of the 11th International Conference on Concurrency Theory (Concur'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 153–167, Pennsylvania, USA, 2000. Springer-Verlag.

[10] E. M. Hahn, G. Li, S. Schewe, A. Turrini, and L. Zhang. Lazy probabilistic model checking without determinisation. In *26th International Conference on Concurrency Theory, CONCUR 2015*, volume 42 of *LIPIcs*, pages 354–367. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[11] J. Holeček, T. Kratochvíla, V. Řehák, D. Šafránek, and P. Šimeček. Verification results in Liberouter project. Technical Report 03, 32pp, CESNET, September 2004.

[12] D. Kini and M. Viswanathan. Limit deterministic and probabilistic automata for LTL$\setminus$GU. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015*, volume 9035 of *LNCS*, pages 628–642. Springer, 2015.

[13] R. Pelánek. BEEM: benchmarks for explicit model checkers. In *Proceedings of the 14th international SPIN conference on Model checking software*, Lecture Notes in Computer Science, pages 263–267. Springer-Verlag, 2007.

[14] S. Safra. On the complexity of omega-automata. In *29th Annual Symposium on Foundations of Computer Science (FOCS'88)*, pages 319–327. IEEE Computer Society, 1988.

[15] S. Sickert, J. Esparza, S. Jaax, and J. Křetínský. Limit-deterministic Büchi automata for linear temporal logic. In *Computer Aided Verification - 28th International Conference, CAV 2016*, volume 9780 of *LNCS*, pages 312–332. Springer, 2016.

[16] S. Sickert and J. Křetínský. Mochiba: Probabilistic LTL model checking using limit-deterministic Büchi automata. In *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016*, volume 9938 of *LNCS*, pages 130–137, 2016.

[17] F. Somenzi and R. Bloem. Efficient Büchi automata for LTL formulæ. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 247–263, Chicago, Illinois, USA, 2000. Springer-Verlag.

[18] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE Computer Society, 1985.

[19] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Computer Society Press, 1986.