From LTL and Limit-Deterministic Büchi Automata to Deterministic Parity Automata

Javier Esparza[1], Jan Křetínský[1(✉)], Jean-François Raskin[2],
and Salomon Sickert[1]

[1] Technische Universität München, Munich, Germany
{esparza,jan.kretinsky,sickert}@in.tum.de
[2] Université libre de Bruxelles, Brussels, Belgium
jraskin@ulb.ac.be

Abstract. Controller synthesis for general linear temporal logic (LTL) objectives is a challenging task. The standard approach involves translating the LTL objective into a deterministic parity automaton (DPA) by means of the Safra-Piterman construction. One of the challenges is the size of the DPA, which often grows very fast in practice, and can reach double exponential size in the length of the LTL formula. In this paper we describe a single exponential translation from limit-deterministic Büchi automata (LDBA) to DPA, and show that it can be concatenated with a recent efficient translation from LTL to LDBA to yield a double exponential, "Safraless" LTL-to-DPA construction. We also report on an implementation, a comparison with the SPOT library, and performance on several sets of formulas, including instances from the 2016 SyntComp competition.

1 Introduction

Limit-deterministic Büchi automata (LDBA, also known as semi-deterministic Büchi automata) were introduced by Courcoubetis and Yannakakis (based on previous work by Vardi) to solve the qualitative probabilistic model-checking problem: Decide if the executions of a Markov chain or Markov Decision Process satisfy a given LTL formula with probability 1 [Var85,VW86,CY95]. The problem faced by these authors was that fully nondeterministic Büchi automata (NBAs), which are as expressible as LTL, and more, cannot be used for probabilistic model checking, and deterministic Büchi automata (DBA) are less expressive than LTL. The solution was to introduce LDBAs as a model in-between: as expressive as NBAs, but deterministic enough.

After these papers, LDBAs received little attention. The alternative path of translating the LTL formula into an equivalent fully deterministic Rabin

automaton using Safra's construction [Saf88] was considered a better option, mostly because it also solves the quantitative probabilistic model-checking problem (computing the probability of the executions that satisfy a formula). However, recent papers have shown that LDBAs were unjustly forgotten. Blahoudek *et al.* have shown that LDBAs are easy to complement [BHS+16]. Kini and Viswanathan have given a single exponential translation of $LTL_{\backslash \mathbf{GU}}$ to LDBA [KV15]. Finally, Sickert *et al.* describe in [SEJK16] a double exponential translation for full LTL that can also be applied to the quantitative case, and behaves better than Safra's construction in practice.

In this paper we add to this trend by showing that LDBAs are also attractive for synthesis. The standard solution to the synthesis problem with LTL objectives consists of translating the LTL formula into a deterministic parity automaton (DPA) with the help of the Safra-Piterman construction [Pit07]. While limit-determinism is not "deterministic enough" for the synthesis problem, we introduce a conceptually simple and worst-case optimal translation LDBA→DPA. Our translation bears some similarities with that of [Fin15] where, however, a Muller acceptance condition is used. This condition can also be phrased as a Rabin condition, but not as a parity condition. Moreover, the way of tracking all possible states and finite runs differs.

Together with the translation LTL→LDBA of [SEJK16], our construction provides a "Safraless", procedure to obtain a DPA from an LTL formula. However, the direct concatenation of the two constructions does not yield an algorithm of optimal complexity: the LTL→LDBA translation is double exponential (and there is a double-exponential lower bound), and so for the LTL→DPA translation we only obtain a triple exponential bound. In the second part of the paper we solve this problem. We show that the LDBAs derived from LTL formulas satisfy a special property, and prove that for such automata the concatenation of the two constructions remains double exponential. To the best of our knowledge, this is the first double exponential "Safraless" LTL→DPA procedure. (Another asymptotically optimal "Safraless" procedure for determinization of Büchi automata with Rabin automata as target has been presented in [FKVW15].)

In the third and final part, we report on the performance of an implementation of our LTL→LDBA→DPA construction, and compare it with algorithms implemented in the SPOT library [DLLF+16]. Note that it is not possible to force SPOT to always produce DPA, sometimes it produces a deterministic generalized Büchi automaton (DGBA). The reason is that DGBA are often smaller than DPA (if they exist) and game-solving algorithms for DGBA are not less efficient than for DPA. Therefore, also our implementation may produce DGBA in some cases. We show that our implementation outperforms SPOT for several sets of parametric formulas and formulas used in synthesis examples taken from the SyntComp 2016 competition, and remains competitive for randomly generated formulas.

Structure of the Paper. Section 2 introduces the necessary preliminaries about automata. Section 3 defines the translation LDBA→DPA. Section 4 shows

how to compose of LTL→LDBA and LDBA→DPA in such a way that the resulting DPA is at most doubly exponential in the size of the LTL formula. Section 5 reports on the experimental evaluation of this worst-case optimal translation, and Sect. 6 contains our conclusions. Several proofs and more details on the implementation can be found in [EKRS17].

2   Preliminaries

Büchi Automata. A (nondeterministic) ω-word automaton A with Büchi acceptance condition (NBA) is a tuple $(Q, q_0, \Sigma, \delta, \alpha)$ where Q is a finite set of states, $q_0 \in Q$ is the *initial* state, Σ is a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $\alpha \subseteq \delta$ is the set of *accepting* transitions[1]. W.l.o.g. we assume that δ is total in the following sense: for all $q \in Q$, for all $\sigma \in \Sigma$, there exists $q' \in Q$ such that $(q, \sigma, q') \in \delta$. A is *deterministic* if for all $q \in Q$, for all $\sigma \in \Sigma$, there exists a unique $q' \in Q$ such that $(q, \sigma, q') \in \delta$. When δ is deterministic and total, it can be equivalently seen as a function $\delta : Q \times \Sigma \to Q$. Given $S \subseteq Q$ and $\sigma \in \Sigma$, let $\mathsf{post}_\delta^\sigma(S) = \{q' \mid \exists q \in S \cdot (q, \sigma, q') \in \delta\}$.

A *run* of A on a ω-word $w : \mathbb{N} \to \Sigma$ is a ω-sequence of states $\rho : \mathbb{N} \to Q$ such that $\rho(0) = q_0$ and for all positions $i \in \mathbb{N}$, we have that $(\rho(i), w(i), \rho(i+1)) \in \delta$. A run ρ is *accepting* if there are infinitely many positions $i \in \mathbb{N}$ such that $(\rho(i), w(i), \rho(i+1)) \in \alpha$. The *language* defined by A, denoted by $\mathsf{L}(A)$, is the set of ω-words w for which A has an accepting run.

A *limit-deterministic Büchi automaton* (LDBA) is a Büchi automaton $A = (Q, q_0, \Sigma, \delta, \alpha)$ such that there exists a subset $Q_d \subseteq Q$ satisfying the three following properties:

1. $\alpha \subseteq Q_d \times \Sigma \times Q_d$, i.e. all accepting transitions are transitions within Q_d;
2. $\forall q \in Q_d \cdot \forall \sigma \in \Sigma \cdot \forall q_1, q_2 \in Q \cdot (q, \sigma, q_1) \in \delta \wedge (q, \sigma, q_2) \in \delta \to q_1 = q_2$, i.e. the transition relation δ is deterministic within Q_d;
3. $\forall q \in Q_d \cdot \forall \sigma \in \Sigma \cdot \forall q' \in Q \cdot (q, \sigma, q') \in \delta \to q' \in Q_d$, i.e. Q_d is a trap (when Q_d is entered it is never left).

W.l.o.g. we assume that $q_0 \in Q \setminus Q_d$, and we denote $Q \setminus Q_d$ by $\overline{Q_d}$. Courcoubetis and Yannakakis show that for every ω-regular language $\mathcal{L}$, there exists an LDBA A such that $\mathsf{L}(A) = \mathcal{L}$ [CY95]. That is, LDBAs are as expressive as NBAs. An example of LDBA is given in Fig. 1. Note that the language accepted by this LDBA cannot be recognized by a deterministic Büchi automaton.

Parity Automata. A deterministic ω-word automaton A with *parity* acceptance condition (DPA) is a tuple $(Q, q_0, \Sigma, \delta, p)$, defined as for deterministic Büchi automata with the exception of the acceptance condition p, which is now

---

[1] Here, we consider automata on infinite words with acceptance conditions based on transitions. It is well known that there are linear translations from automata with acceptance conditions defined on transitions to automata with acceptance conditions defined on states, and vice-versa.
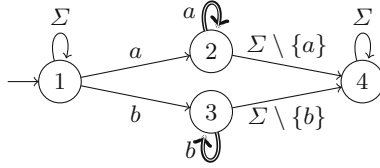
Fig. 1. An LDBA for the LTL language $\mathbf{FG}a \vee \mathbf{FG}b$. The behavior of A is deterministic within the subset of states $Q_d = \{2, 3, 4\}$ which is a trap, the set of accepting transitions are depicted in bold face and they are defined only between states of Q_d.

a function assigning an integer in $\{1, 2, \ldots, d\}$, called a *color*, to each transition in the automaton. Colors are naturally ordered by the order on integers.

Given a run ρ over a word w, the infinite sequence of colors traversed by the run ρ is noted $p(\rho)$ and is equal to $p(\rho(0), w(0), \rho(1)) \ p((\rho(1), w(1), \rho(2))) \ldots p(\rho(n), w(n), \rho(n+1)) \ldots$. A run ρ is *accepting* if the minimal color that appears infinitely often along $p(\rho)$ is *even*. The *language* defined by A, denoted by $\mathsf{L}(A)$ is the set of ω-words w for which A has an accepting run.

While deterministic Büchi automata are not expressively complete for the class of ω-regular languages, DPAs are complete for ω-regular languages: for every ω-regular language $\mathcal{L}$ there exists a DPA A such that $\mathsf{L}(A) = \mathcal{L}$, see e.g. [Pit07].

3   From LDBA to DPA

3.1   Run DAGs and Their Coloring

Run DAG. A nondeterministic automaton A may have several (even an infinite number of) runs on a given ω-word w. As in [KV01], we represent this set of runs by means of a directed acyclic graph structure called the *run DAG* of A on w. Given an LDBA $A = (Q, Q_d, q_0, \Sigma, \delta, \alpha)$, this graph $G_w = (V, E)$ has a set of vertices $V \subseteq Q \times \mathbb{N}$ and edges $E \subseteq V \times V$ defined as follows:

- $V = \bigcup_{i \in \mathbb{N}} V_i$, where the sets V_i are defined inductively:
  - $V_0 = \{(q_0, 0)\}$, and for all $i \geq 1$,
  - $V_i = \{(q, i) \mid \exists (q', i-1) \in V_{i-1} : (q', w(i), q) \in \delta\}$;
- $E = \{((q, i), (q', i+1)) \in V_i \times V_{i+1} \mid (q, w(i), q') \in \delta\}$.

We denote by V_i^d the set $V_i \cap (Q_d \times \{i\})$ that contains the subset of vertices of layer i that are associated with states in Q_d.

Observe that all the paths of G_w that start from $(q_0, 0)$ are runs of A on w, and, conversely, each run ρ of A on w corresponds exactly to one path in G_w that starts from $(q_0, 0)$. So, we call *runs* the paths in the run DAG G_w. In particular, we say that an infinite path $v_0 v_1 \ldots v_n \ldots$ of G_w is an accepting run if there are infinitely many positions $i \in \mathbb{N}$ such that $v_i = (q, i)$, $v_{i+1} = (q', i+1)$, and $(q, w(i), q') \in \alpha$. Clearly, w is accepted by A if and only if there is an accepting run in G_w. We denote by $\rho(0..n) = v_0 v_1 \ldots v_n$ the prefix of length $n+1$ of the run ρ.

Ordering of Runs. A function $\mathsf{Ord} : Q \to \{1, 2, \ldots, |Q_d|, +\infty\}$ is called an *ordering* of the states of A w.r.t. Q_d if Ord defines a strict total order on the state from Q_d, and maps each state $q \in \overline{Q_d}$ to $+\infty$, i.e.:

- for all $q \in \overline{Q_d}$, $\mathsf{Ord}(q) = +\infty$,
- for all $q \in Q_d$, $\mathsf{Ord}(q) \neq +\infty$, and
- for all $q, q' \in Q_d$, $\mathsf{Ord}(q) = \mathsf{Ord}(q')$ implies $q = q'$.

We extend Ord to vertices in G_w as follows: $\mathsf{Ord}((q, i)) = \mathsf{Ord}(q)$.

Starting from Ord, we define the following pre-order on the set of run prefixes of the run DAG G_w. Let $\rho(0..n) = v_0 v_1 \ldots v_n \ldots$ and $\rho'(0..n) = v_0' v_1' \ldots v_n' \ldots$ be two run prefixes of length $n + 1$, we write $\rho(0..n) \sqsubseteq \rho'(0..n)$, if $\rho(0..n)$ is *smaller than* $\rho'(0..n)$, which is defined as:

- for all i, $0 \leq i \leq n$, $\mathsf{Ord}(\rho(i)) = \mathsf{Ord}(\rho'(i))$, or
- there exists i, $0 \leq i \leq n$, such that:
    - $\mathsf{Ord}(\rho(i)) < \mathsf{Ord}(\rho'(i))$, and
    - for all j, $0 \leq j < i$, $\mathsf{Ord}(\rho(j)) = \mathsf{Ord}(\rho'(j))$.

This is extended to (infinite) runs as: $\rho \sqsubseteq \rho'$ iff for all $i \geq 0 \cdot \mathsf{Ord}(\rho(0..i)) \sqsubseteq \mathsf{Ord}(\rho'(0..i))$.

Remark 1. If A accepts a word w, then A has a $\sqsubseteq$-smallest accepting run for w.

We use the $\sqsubseteq$-relation on run prefixes to order the vertices of V_i that belong to Q_d: for two different vertices $v = (q, i) \in V_i$ and $v' = (q', i) \in V_i$, v is $\sqsubset_i$-smaller than v', if there is a run prefix of G_w that ends up in v which is $\sqsubseteq$-smaller than all the run prefixes that ends up in v', which induces a total order among the vertices of V_i^d because the states in Q_d are totally ordered by the function Ord.

Lemma 1. *For all $i \geq 0$, for two different vertices $v = (q, i), v' = (q', i) \in V_i^d$, then either $v \sqsubset_i v'$ or $v' \sqsubset_i v$, i.e., $\sqsubset_i$ is a total order on V_i^d.*

Indexing Vertices. The index of a vertex $v = (q, i) \in V_i$ such that $q \in Q_d$, denoted by $\mathsf{Ind}_i(v)$, is a value in $\{1, 2, \ldots, |Q_d|\}$ that denotes its order in V_i^d according to $\sqsubset_i$ (the $\sqsubset_i$-smallest element has index 1). For $i \geq 0$, we identify two important sets of vertices:

- $\mathsf{Dec}(V_i^d)$ is the set of vertices $v \in V_i^d$ such that there exists a vertex $v' \in V_{i+1}^d$: $(v, v') \in E$ and $\mathsf{Ind}_{i+1}(v') < \mathsf{Ind}_i(v)$, i.e. the set of vertices in V_i^d whose (unique) successor in V_{i+1}^d has a smaller index value.
- $\mathsf{Acc}(V_i^d)$ is the set of vertices $v = (q, i) \in V_i^d$ such that there exists $v' = (q', i+1) \in V_{i+1}^d$: $(v, v') \in E$ and $(q, w(i), q') \in \alpha$, i.e. the set of vertices in V_i^d that are the source of an accepting transition on $w(i)$.

Remark 2. Along a run, the index of vertices can only decrease. As the function $\mathsf{Ind}(\cdot)$ has a finite range, the index along a run has to eventually stabilize.

Assigning Colors. The set of colors that are used for coloring the levels of the run DAG G_w is $\{1, 2, \ldots, 2 \cdot |Q_d| + 1\}$. We associate a color with each transition from level i to level $i + 1$ according to the following set of cases:

1. if $\mathsf{Dec}(V_i^d) = \emptyset$ and $\mathsf{Acc}(V_i^d) \neq \emptyset$, the color is $2 \cdot \min_{v \in \mathsf{Acc}(V_i^d)} \mathsf{Ind}_i(v)$.
2. if $\mathsf{Dec}(V_i^d) \neq \emptyset$ and $\mathsf{Acc}(V_i^d) = \emptyset$, the color is $2 \cdot \min_{v \in \mathsf{Dec}(V_i^d)} \mathsf{Ind}_i(v) - 1$.
3. if $\mathsf{Dec}(V_i^d) \neq \emptyset$ and $\mathsf{Acc}(V_i^d) \neq \emptyset$, the color is defined as the minimal color among
   - $c_{\mathsf{odd}} = 2 \cdot \min_{v \in \mathsf{Dec}(V_i^d)} \mathsf{Ind}_i(v) - 1$, and
   - $c_{\mathsf{even}} = 2 \cdot \min_{v \in \mathsf{Acc}(V_i^d)} \mathsf{Ind}_i(v)$.
4. if $\mathsf{Dec}(V_i^d) = \mathsf{Acc}(V_i^d) = \emptyset$, the color is $2 \cdot |Q_q| + 1$.

The intuition behind this coloring is as follows: the coloring tracks runs in Q_d (only those are potentially accepting as $\alpha \subseteq Q_d \times \Sigma \times Q_d$) and tries to produce an even color that corresponds to the smallest index of an accepting run. If in level i the run DAG has an outgoing transition that is accepting, then this is a *positive event*, as a consequence the color emitted is *even* and it is a function of the smallest index of a vertex associated with an accepting transition from V_i to V_{i+1}. Runs in Q_d are deterministic but they can merge with *smaller* runs. When this happens, this is considered as a *negative event* because the even colors that have been emitted by the run that merges with the smaller run should not be taken into account anymore. As a consequence an odd color is emitted in order to cancel all the (good) even colors that were generated by the run that merges with the smaller one. In that case the odd color is function of the smallest index of a run vertex in V_i whose run merges with a smaller vertex in V_{i+1}. Those two first cases are handled by cases 1 and 2 of the case study above. When both situations happen at the same time, then the color is determined by the minimum of the two colors assigned to the positive and the negative events. This is handled by case 3 above. And finally, when there is no accepting transition from V_i to V_{i+1} and no merging, the largest odd color is emitted as indicated by case 4 above.

According to this intuition, we define the *color summary* of the run DAG G_w as the minimal color that appears infinitely often along the transitions between its levels. Because of the deterministic behavior of the automaton in Q_d, each run can only merge at most $|Q_d| - 1$ times with a smaller one (the size of the range of the function $\mathsf{Ind}(\cdot)$ minus one), and as a consequence of the definition of the above coloring, we know that, on word accepted by A, the smallest accepting run will eventually generate infinitely many (good) even colors that are never trumped by smaller odd colors.

Example 1. The left part of Fig. 2 depicts the run DAG of the limit-deterministic automaton of Fig. 1 on the word $w = abb(ab)^\omega$. Each path in this graph represents a run of the automaton on this word. The coloring of the run DAG follows the
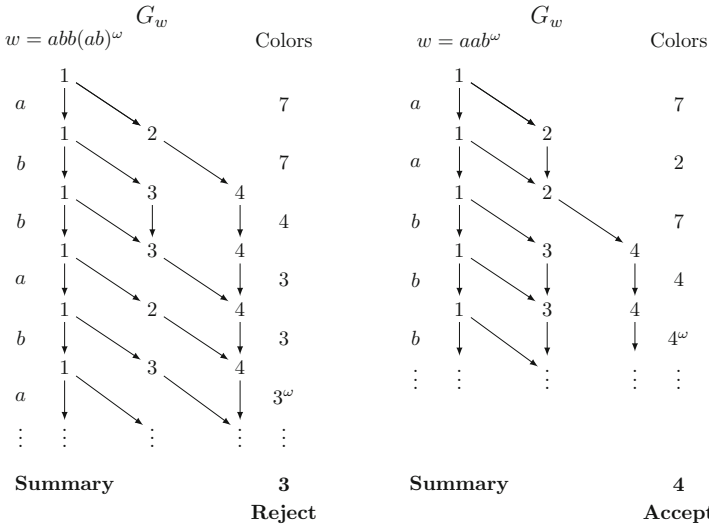
Fig. 2. The run DAGs automaton of Fig. 1 on the word $w = (ab)^\omega$ given on the left, and on the word $w = aab^\omega$ given on the right, together with their colorings.

coloring rules defined above. Between level 0 and level 1, the color is equal to $7 = 2|Q_d| + 1$, as no accepting edge is taken from level 0 to level 1 and no run merges (within Q_d). The color 7 is also emitted from level 1 to level 2 for the same reason. The color 4 is emitted from level 2 to level 3 because the accepting edge $(3, b, 3)$ is taken and the index of state 3 in level 2 is equal to 2 (state 4 has index 1 as it is the end point of the smallest run prefix within Q_d). The color 3 is emitted from level 3 to level 4 because the run that goes from 3 to 4 merges with the smaller run that goes from 4 to 4. In order to cancel the even colors emitted by the run that goes from 3 to 4, color 3 is emitted. It cancels the even color 4 emitted before by this run. Afterwards, colors 3 is emitted forever. The color summary is 3 showing that there is no accepting run in the run DAG.

The right part of Fig. 2 depicts the run DAG of the limit deterministic automaton of Fig. 1 on the word $w = aab^\omega$. The coloring of the run DAG follows the coloring rules defined above. Between levels 0 and 1, color 7 is emitted because no accepting edge is crossed. To the next level, we see the accepting edge $(2, a, 2)$ and color $2 \cdot 1 = 2$ is emitted. Upon reading the first b, we see again 7 since there is neither any accepting edge seen nor any merging takes place. Afterwards, each b causes an accepting edge $(3, b, 3)$ to be taken. While the smallest run, which visits 4 forever, is not accepting, the second smallest run that visits 3 forever is accepting. As 3 has index 2 in all the levels below level 3, the color is forever equal to 4. The color summary of the run is thus equal to $2 \cdot 2 = 4$ and this shows that word $w = aab^\omega$ is accepted by our limit deterministic automaton of Fig. 1.

The following theorem tells us that the color summary (the minimal color that appears infinitely often) can be used to identify run DAGs that contain accepting runs. The proof can be found in [EKRS17, Appendix A].

Theorem 1. *The color summary of the run DAG G_w is even if and only if there is an accepting run in G_w.*

3.2   Construction of the DPA

From an LDBA $A = (Q, Q_d, q_0, \Sigma, \delta, \alpha)$ and an ordering function $\mathsf{Ord} : Q \to \{1, 2, \ldots, |Q_d|, +\infty\}$ compatible with Q_d, we construct a deterministic parity automaton $B = (Q^B, q_0^B, \Sigma, \delta^B, p)$ that, on a word w, constructs the levels of the run DAG G_w and the coloring of previous section. Theorem 1 tells us that such an automaton accepts the same language as A.

First, we need some notations. Given a finite set S, we note $\mathcal{P}(S)$ the set of its subsets, and $\mathcal{OP}(S)$ the set of its totally ordered subsets. So if $(s, <) \in \mathcal{OP}(S)$ then $s \subseteq S$ and $< \subseteq s \times s$ is a total strict order on s. For $e \in s$, we denote by $\mathsf{Ind}_{(s,<)}(e)$ the position of $e \in s$ among the elements in s for the total strict order $<$, with the convention that the index of the $<$-minimum element is equal to 1. The deterministic parity automaton $B = (Q^B, q_0^B, \Sigma, \delta^B, p)$ is defined as follows.

States and Initial State. The set of states is $Q^B = \mathcal{P}(\overline{Q_d}) \times \mathcal{OP}(Q_d)$, i.e. a state of B is a pair $(s, (t, <))$ where s is a set of states outside Q_d, and t is an ordered subset of Q_d. The ordering reflects the relative index of each state within t. The initial state is $q_0^B = (\{q_0\}, (\{\}, \{\}))$.

Transition Function. Let $(s_1, (t_1, <_1))$ be a state in Q^B, and $\sigma \in \Sigma$. Then $\delta^B((s_1, (t_1, <_1))) = (s_2, (t_2, <_2))$ where:

- $s_2 = \mathsf{post}_\delta^\sigma(s_1) \cap \overline{Q_d}$;
- $t_2 = \mathsf{post}_\delta^\sigma(s_1 \cup t_1) \cap Q_d$;
- $<_2$ is defined from $<_1$ and Ord as follows: $\forall q_1, q_2 \in t_2 \colon q_1 <_2 q_2$ iff:
    1. **either**, $\neg \exists q_1' \in t_1 : q_1 = \delta(q_1', \sigma)$, and $\neg \exists q_2' \in t_1 : q_2 = \delta(q_2', \sigma)$, and $\mathsf{Ord}(q_1) < \mathsf{Ord}(q_2)$,
       i.e. none has a predecessor in Q_d, then they are ordered using Ord;
    2. **or**, $\exists q_1' \in t_1 : q_1 = \delta(q_1', \sigma)$, and $\neg \exists q_2' \in t_1 : q_2 = \delta(q_2', \sigma)$,
       i.e. q_1 has a σ-predecessor in Q_d, and q_2 not;
    3. **or** $\exists q_1' \in t_1 : q_1 = \delta(q_1', \sigma)$, and $\exists q_2' \in t_1 : q_2 = \delta(q_2', \sigma)$, and $\min_{<_1}\{q_1' \in t_1 \mid q_1 = \delta(q_1', \sigma)\} < \min_{<_1}\{q_2' \in t_1 \mid q_2 = \delta(q_2', \sigma)\}$,
       i.e. both have a predecessor in Q_d, and they are ordered according to the order of their minimal parents.

Coloring. To define the coloring of edges in the deterministic automaton, we need to identify the states $q \in t_1$ in a transition $(s_1, (t_1, <_1)) \xrightarrow{\sigma} (s_2, (t_2, <_2))$ whose indices decrease when going from t_1 to t_2. Those are defined as follows:

$$\mathsf{Dec}(t_1) = \{q_1 \in t_1 \mid \mathsf{Ind}_{(t_2, <_2)}(\delta(q_1, \sigma)) < \mathsf{Ind}_{(t_1, <_1)}(q_1)\}.$$

Additionally, let $\mathsf{Acc}(t_1) = \{q \mid \exists q' \in t_2 : (q, \sigma, q') \in \alpha\}$ denote the subset of states in t_1 that are the source of an accepting transition.

We assign a color to each transition $(s_1, (t_1, <_1)) \to^\sigma (s_2, (t_2, <_2))$ as follows:

1. if $\mathsf{Dec}(t_1) = \emptyset$ and $\mathsf{Acc}(t_1) \neq \emptyset$, the color is $2 \cdot \min_{q \in \mathsf{Acc}(t_1)} \mathsf{Ind}_{(t_1, <_1)}(q)$.
2. if $\mathsf{Dec}(t_1) \neq \emptyset$ and $\mathsf{Acc}(t_1) = \emptyset$, the color is $2 \cdot \min_{q \in \mathsf{Dec}(t_1)} \mathsf{Ind}_{(t_1, <_1)}(q) - 1$.
3. if $\mathsf{Dec}(t_1) \neq \emptyset$ and $\mathsf{Acc}(t_1) \neq \emptyset$, the color is defined as the minimal color among
   - $c_{\mathsf{odd}} = 2 \cdot \min_{q \in \mathsf{Dec}(t_1)} \mathsf{Ind}_{(t_1, <_1)}(q) - 1$, and
   - $c_{\mathsf{even}} = 2 \cdot \min_{q \in \mathsf{Acc}(t_1)} \mathsf{Ind}_{(t_1, <_1)}(q)$.
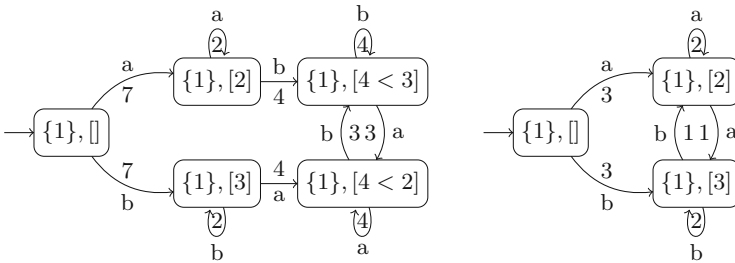4. if $\mathsf{Dec}(t_1) = \mathsf{Acc}(t_1) = \emptyset$, the color is $2 \cdot |Q_q| + 1$.

Fig. 3. Left: DPA that accepts the LTL language $\mathbf{FG}a \vee \mathbf{FG}b$, edges are decorated with a natural number that specifies its color. Right: A reduced DPA.

Example 2. The DPA of Fig. 3 is the automaton that is obtained by applying the construction LDBA→DPA defined above to the LDBA of Fig. 1 that recognizes the LTL language $\mathbf{FG}a \vee \mathbf{FG}b$. The figure only shows the reachable states of this construction. As specified in the construction above, states of DPA are labelled with a subset of $\overline{Q_d}$ and a ordered subset of Q_d of the original NBA. As an illustration of the definitions above, let us explain the color of edges from state $(\{1\}, [4, 3])$ to itself on letter b. When the NBA is in state 1, 3 or 4 and letter b is read, then the next state of the automaton is again 1, 3 or 4. Note also that there are no runs that are merging in that case. As a consequence, the color that is emitted is even and equal to the index of the smallest state that is the target of an accepting transition. In this case, this is state 3 and its index is 2. This is the justification for the color 4 on the edge. On the other hand, if letter a is read from state $(\{1\}, [4, 3])$, then the automaton moves to states $(\{1\}, [4, 2])$. The state 3 is mapped to state 4 and there is a run merging which induces that the color emitted is odd and equal to 3. This 3 trumps all the 4's that were possibly emitted from state $(\{1\}, [4, 3])$ before.

Theorem 2. *The language defined by the deterministic parity automaton B is equal to the language defined by the limit deterministic automaton A, i.e.* $\mathsf{L}(A) = \mathsf{L}(B)$.

Proof. Let $w \in \Sigma^\omega$ and G_w be the run DAG of A on w. It is easy to show by induction that the sequence of colors that occur along G_w is equal to the sequence of colors defined by the run of the automaton B on w. By Theorem 1, the language of automaton B is thus equal to the language of automaton A. □

3.3   Complexity Analysis

Upper Bound. Let $n = |Q|$ be the size of the LDBA and let $n_d = |Q_d|$ be the size of the accepting component. We can bound the number of different orderings using the series of reciprocals of factorials (with e being Euler's number):

$$|\mathcal{OP}(Q_d)| = \sum_{i=0}^{n_d} \frac{n_d!}{(n_d - i)!} \leq n_d \cdot n_d! \cdot \sum_{i=0}^{\infty} \frac{1}{i!} = e \cdot n_d \cdot n_d! \in \mathcal{O}(2^{n \cdot \log n})$$

Thus the obtained DPA has $\mathcal{O}(2^n \cdot 2^{n \cdot \log n}) = 2^{\mathcal{O}(n \cdot \log n)}$ states and $\mathcal{O}(n)$ colours.

Lower Bound. We obtain a matching lower bound by strengthening Theorem 8 from [Löd99]:

Lemma 2. *There exists a family $(L_n)_{n \geq 2}$ of languages (L_n over an alphabet of n letters) such that for every n the language L_n can be recognized by a limit-deterministic Büchi automaton with $3n + 2$ states but can not be recognized by a deterministic Parity automaton with less than $n!$ states.*

Proof. The proof of Theorem 8 from [Löd99] constructs a non-deterministic Büchi automaton of exactly this size and which is in fact limit-deterministic.

Assume there exists a deterministic Parity automata for L_n with $m < n!$ states. Since parity automata are closed under complementation, we can obtain a parity automaton and hence also a Rabin automaton of size m for $\overline{L_n}$ and thus a Streett automaton of size m for L_n, a contradiction to Theorem 8 of [Löd99]. □

Corollary 1. *Every translation from limit-deterministic Büchi automata of size n to deterministic parity yields automata with $2^{\Omega(n \log n)}$ states in the worst case.*

4   From LTL to Parity in $2^{2^{\mathcal{O}(n)}}$

In [SEJK16] we present a LTL→LDBA translation. Given a formula φ of size n, the translation produces an asymptotically optimal LDBA with $2^{2^{\mathcal{O}(n)}}$ states. The straightforward composition of this translation with the single exponential LDBA→DPA translation of the previous section is only guaranteed to be triple exponential, while the Safra-Piterman construction produces a DPA of at most doubly exponential size. In this section we describe a modified composition that yields a double exponential DPA. To the best of our knowledge this is the first

translation of the whole LTL to deterministic parity automata that is asymp-
totically optimal and does not use Safra's construction.

The section is divided into two parts. In the first part, we explain and illus-
trate a redundancy occurring in our LDBA→DPA translation, responsible for
the undesired extra exponential. We also describe an optimization that removes
this redundancy when the LDBA satisfies some conditions. In the second part,
we show these conditions are satisfied on the products of the LTL→LDBA trans-
lation, which in turn guarantees a doubly exponential LTL→DPA procedure.

4.1 An Improved Construction

We can view the second component of a state of the DPA as a sequence of states
of the LDBA, ordered by their indices. Since there are $2^{2^{\mathcal{O}(n)}}$ states of the LDBA
for an LTL formula of length n, the number of such sequences is

$$2^{2^{\mathcal{O}(n)}}! = 2^{2^{2^{\mathcal{O}(n)}}}$$

If only the length of the sequences (the maximum index) were bounded by 2^n,
the number of such sequences would be smaller than the number of functions
$2^n \to 2^{2^{\mathcal{O}(n)}}$ which is

$$(2^{2^{\mathcal{O}(n)}})^{2^n} = 2^{2^{\mathcal{O}(n)} \cdot 2^n} = 2^{2^{\mathcal{O}(n)}}$$

Fix an LDBA with set of states Q. Assume the existence of an *oracle*: a list
of statements of the form $\mathsf{L}(q) \subseteq \bigcup_{q' \in Q_q} \mathsf{L}(q')$ where $q \in Q$ and $Q_q \subseteq Q$. We use
the oracle to define a mapping that associates to each run DAG G_w a "reduced
DAG" G_w^*, defined as the result of iteratively performing the following four-step
operation:

– Find the first V_i in the current DAG such that the sequence $(v_1, i) \sqsubset (v_2, i) \sqsubset
\cdots \sqsubset (v_{n_i}, i)$ of vertices of V_i^d contains a vertex (v_k, i) for which the oracle
ensures

$$\mathsf{L}(v_k) \subseteq \bigcup_{j < k} \mathsf{L}(v_j) \qquad (*)$$

We call (v_k, i) a *redundant vertex*.
– Remove (v_k, i) from the sequence, and otherwise keep the ordering $\sqsubseteq_i$
unchanged (thus decreasing the index of vertices (v, ℓ) with $\ell > k$).
– Redirect transitions leading from vertices in V_{i-1} to (v_k, i) so that they lead
to the smallest vertex (v_1, i) of V_i.
– Remove any vertices (if any) that are no longer reachable from vertices of V_1.

We define the color summary of G_w^* in exactly the same way as the color summary
of G_w. The DAG G_w^* satisfies the following crucial property, whose proof can be
found in [EKRS17, Appendix B]:

Proposition 1. *The color summary of the run DAG G_w^* is even if and only if there is an accepting run in G_w.*

The mapping on DAGs induces a reduced DPA as follows. The states are the pairs $(s, (t, <))$ such that $(t, <)$ does not contain redundant vertices. There is a transition $(s_1, (t_1, <)) \xrightarrow{a} (s_2, (t_2, <))$ with color c iff there is a word w and an index i such that $(s_1, (t_1, <))$ and $(s_2, (t_2, <))$ correspond to the i-th and $(i+1)$-th levels of G_w^*, and a and c are the letter and color of the step between these levels in G_w^*. Observe that the set of transitions is independent of the words chosen to define them.

The equivalence between the initial DPA $\mathcal{A}$ and the reduced DPA $\mathcal{A}_r$ follows immediately from Proposition 1: $\mathcal{A}$ accepts w iff G_w contains an accepting run iff the color summary of G_w^* is even iff $\mathcal{A}_r$ accepts w.

Example 3. Consider the LDBA of Fig. 1 and an oracle given by $\mathsf{L}(4) = \emptyset$, ensuring $\mathsf{L}(4) \subseteq \bigcup_{i \in I} \mathsf{L}(i)$ for any $I \subseteq Q$. Then 4 is always redundant and merged, removing the two rightmost states of the DPA of Fig. 3(left), resulting in the DPA of Fig. 3(right). However, for the sake of technical convenience, we shall refrain from removing a redundant vertex when it is the smallest one (with index 1).

Since the construction of the reduced DPA is parametrized by an oracle, the obvious question is how to obtain an oracle that does not involve applying an expensive language inclusion test. Let us give a first example in which an oracle can be easily obtained:

Example 4. Consider an LDBA where each state $v = \{s_1, \ldots, s_k\}$ arose from some powerset construction on an NBA in such a way that $\mathsf{L}(\{s_1, \ldots, s_k\}) = \mathsf{L}(s_1) \cup \cdots \mathsf{L}(s_k)$. An oracle can, for instance, allow us to merge whenever $v_k \subseteq \bigcup_{j<k} v_j$, which is a sound syntactic approximation of language inclusion. This motivates the following formal generalization.

Let $\mathcal{L}_B = \{L_i \mid i \in B\}$ be a finite set of languages, called *base* languages. We call $\mathcal{L}_C := \{\bigcup \mathcal{L} \mid \mathcal{L} \subseteq \mathcal{L}_B\}$ the join-semilattice of *composed* languages. We shall assume an LDBA with some $\mathcal{L}_B$ such that $\mathsf{L}(q) \in \mathcal{L}_C$ for every state q. We say that such an LDBA *has a base* $\mathcal{L}_B$. In other words, every state recognizes a union of some base languages. (Note that every automaton has a base of at most linear size.) Whenever we have states v_j recognizing $\bigcup_{i \in I_j} L_i$ with $I_j \subseteq B$ for every j, the oracle allows us to merge vertices v_k satisfying $I_k \subseteq \bigcup_{j<k} I_j$. Intuitively, the oracle declares a vertex redundant whenever the simple syntactic check on the indices allows for that.

Let $V_1 = \bigcup_{i \in I_1} L_i, \cdots V_j = \bigcup_{i \in I_j} L_i$ be a sequence of languages of $\mathcal{L}_C$ where the reduction has been applied and there are no more redundant vertices. The maximum length of such a sequence is given already by the base $\mathcal{L}_B$ and we denote it $width(\mathcal{L}_B)$.

Lemma 3. *For any $\mathcal{L}_B$, we have $width(\mathcal{L}_B) \leq |\mathcal{L}_B| + 1$.*

Proof. We provide an injective mapping of languages in the sequence (except for V_1) into B. Since $I_2 \not\subseteq I_1$, there is some $i \in I_2 \setminus I_1$ and we map V_2 to this i. In general, since $I_k \not\subseteq \bigcup_{j=1}^{k-1} I_j$, we also have $i \in I_k \setminus \bigcup_{j=1}^{k-1} I_j$ and we map V_k to this i.   □

On the one hand, the transformation of LDBA to DPA without the reduction yields $2^{\mathcal{O}(|Q| \cdot \log |Q|)}$ states. On the other hand, we can now show that the second component of reduced LDBA with a base can be exponentially smaller. Further, let us assume the LDBA is *initial-deterministic*, meaning that $\delta \cap (\overline{Q_d} \times \Sigma \times \overline{Q_d})$ is deterministic, thus not resulting in blowup in the first component.

Corollary 2. *For every initial-deterministic LDBA with base of size m, there is an equivalent DPA with $2^{\mathcal{O}(m^2)}$ states.*

Proof. The number of composed languages is $\mathcal{L}_C = 2^m$. Therefore, the LDBA has at most 2^m (non-equivalent) states. Hence the construction produces at most

$$|\mathcal{L}_C| \cdot |\mathcal{L}_C|^{\mathcal{O}(width(\mathcal{L}_B))} = 2^m \cdot (2^m)^{\mathcal{O}(m)} = 2^{\mathcal{O}(m^2)}$$

states since the LDBA is initial-deterministic, causing no blowup in the first component.   □

4.2   Bases for LDBAs Obtained from LTL Formulas

We prove that the width for LDBA arising from the LTL transformation is only singly exponential in the formula size. To this end, we need to recall a property of the LTL→LDBA translation of [SEJK16]. Since partial evaluation of formulas plays a major role in the translation, we introduce the following definition. Given an LTL formula φ and sets T and F of LTL formulas, let $\varphi[T, F]$ denote the result of substituting **tt** (true) for each occurrence of a formula of T in φ, and similarly **ff** (false) for formulas of F. The following property of the translation is proven in [EKRS17, Appendix C].

Proposition 2. *For every LTL formula φ, every state s of the LDBA of [SEJK16] is labelled by an LTL formula $label(s)$ such that (i) $\mathsf{L}(s) = \mathsf{L}(label(s))$ and (ii) $label(s)$ is a Boolean combination of subformulas of $\varphi[T_s, F_s]$ for some T_s and F_s. Moreover, the LDBA is initial-deterministic.*

As a consequence, we can bound the corresponding base:

Corollary 3. *For every LTL formula φ, the LDBA of [SEJK16] for φ has a base of size $2^{\mathcal{O}(|\varphi|)}$.*

Proof. Firstly, we focus on states using the same $\varphi[T_s, F_s]$. The language of each state can be defined by a Boolean formula over $\mathcal{O}(|\varphi|)$ atoms. Since every Boolean formula can be expressed in the disjunctive normal form, its language is a union of the conjuncts. The conjunctions thus form a base for these states.

There are exponentially many different conjunction in the number of atoms. Hence the base is of singly exponential size $2^{\mathcal{O}(|\varphi|)}$ as well.

Secondly, observe that there are only $2^{\mathcal{O}(|\varphi|)}$ different formulas $\varphi[T_s, F_s]$ and thus only $2^{\mathcal{O}(|\varphi|)}$ different sets of atoms. Altogether, the size is bounded by

$$2^{\mathcal{O}(|\varphi|)} \cdot 2^{\mathcal{O}(|\varphi|)} = 2^{\mathcal{O}(|\varphi|)}$$

$\square$

Theorem 3. *For every LTL formula φ, there is a DPA with $2^{2^{\mathcal{O}(|\varphi|)}}$ states.*

Proof. The LDBA for φ has base of singly exponential size $2^{\mathcal{O}(|\varphi|)}$ by Corollary 3 and is initial-deterministic by Proposition 2. Therefore, by Corollary 2, the size of the DPA is doubly exponential, in fact

$$2^{\left(2^{\mathcal{O}(|\varphi|)}\right)^2} = 2^{2^{\mathcal{O}(|\varphi|)}}$$

$\square$

This matches the lower bound $2^{2^{\Omega(n)}}$ by [KR10] as well as the upper bound by the Safra-Piterman approach. Finally, note that while the breakpoint constructions in [SEJK16] is analogous to Safra's vertical merging, the merging introduced here is analogous to Safra's horizontal merging.

5   Experimental Evaluation

We evaluate the performance of our construction on several datasets taken from [BKS13,DWDMR08,SEJK16] and several Temporal Logic Synthesis Format (TLSF) specifications [JBB+16] of the SyntComp 2016 competition.

We use the size of the constructed deterministic automaton as an indicator for the overall performance of the synthesis procedure. In [ST03] it is argued that the degree of determinism of the automaton is a better predictor for performance in model-checking problems; however, this parameter is not applicable for synthesis problems, which require deterministic automata.

We compare two versions of our implementation (with and without optimizations, see below) with the algorithms of Spot [DLLF+16]. Each tool is given 64 GB of memory and 10 min. Increasing time to 10 hours does not change the results. More precisely, we compare the following three setups:

S. (`ltl2tgba`, 2.1.1) - Spot [DLLF+16] implements a version of the Safra-Piterman determinization procedure [Red12] with several optimizations.

L2P and L2P′. (`ltl2dpa`, 1.0.0) - L2P is the construction of this paper, available at https://www7.in.tum.de/~sickert/projects/ltl2dpa. L2P′ adds two optimizations. First, the tool translates both the formula and its negation to DPAs A_1, A_2, complements A_2 to yield $\overline{A_2}$, and picks the smaller of A_1, A_2. Further, we apply the simplification routines of Spot (`ltlfilt` and `autfilt`, respectively).

We consider three groups of benachmarks:

(a) Parametrised Formulas

(b) Randomly Generated Formulas

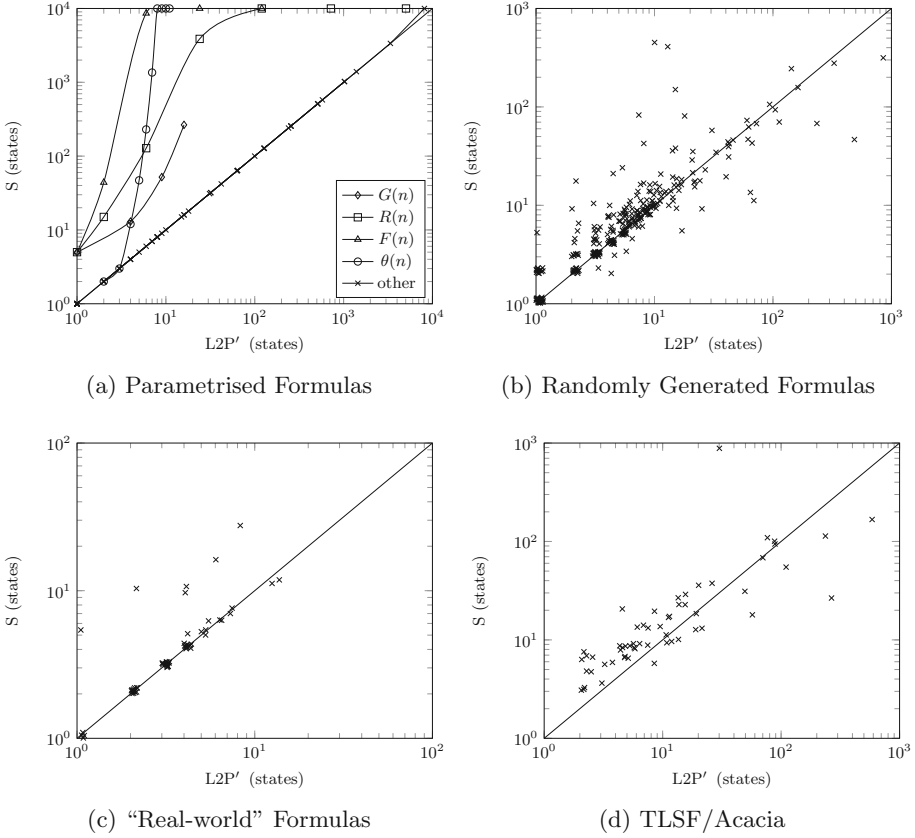(c) "Real-world" Formulas

(d) TLSF/Acacia

Fig. 4. Comparison of Spot and our implementation using the best configurations. Timeouts are denoted by setting the size of the automaton to the maximum.

Parametric Formulas. 10 benchmarks from [BKS13,SEJK16]). In six cases S and L2P′ produce identical results. The other four are

$$R(n) = \bigwedge_{i=1}^{n}(\mathbf{GF}p_i \vee \mathbf{FG}p_{i+1}) \qquad G(n) = (\bigwedge_{i=1}^{n}\mathbf{GF}p_i) \to (\bigwedge_{i=1}^{n}\mathbf{GF}q_i)$$
$$\theta(n) = \neg((\bigwedge_{i=1}^{n}\mathbf{GF}p_i) \to \mathbf{G}(q \to \mathbf{F}r)) \quad F(n) = \bigwedge_{i=1}^{n}(\mathbf{GF}p_i \to \mathbf{GF}q_i)$$

for which the results are shown in (Fig. 4a). Additionally, we consider the "*f*" formulas from [SEJK16] (Table 1). Observe that L2P′ performs clearly better, and the gap between the tools grows when the parameter increases.

Randomly Generated Formulas from [BKS13] (Fig. 4b).

Real Data. Formulas taken from case studies and synthesis competitions—the intended domain of application of our approach. Figure 4c and d show results for the real-world formulas of [BKS13] and the TLSF specifications contained in the Acacia set of [JBB+16]. Table 1 shows results for LTL formulas expressing properties of Szymanski's protocol [DWDMR08], and for the generalised buffer benchmark of Acacia.

Table 1. Number of states and number of used colours in parenthesis for the constructed automata. Timeouts are marked with t.

	$f(1,0)$	$f(1,2)$	$f(1,4)$	$f(2,0)$	$f(2,2)$	zn	zp1	zp2	zp3	Buffer
S	18(6)	141(8)	2062(8)	208(12)	883(12)	t	t	t	t	t
L2P	12(8)	114(9)	332(15)	144(14)	4732(19)	t	t	t	t	1425(27)
L2P'	12(8)	78(7)	271(11)	106(9)	1904(15)	32(6)	42(6)	111(12)	97(12)	435(4)

Average Compression Ratios. The geometric average compression ratio for a benchmark suite B is defined as $\prod_{\varphi \in B}(n_\varphi^S/n_\varphi^{L2P'})^{1/|B|}$, where n_φ^S and $n_\varphi^{L2P'}$ denote the number of states of the automata produced by Spot and L2P', respectively. The ratios in our experiments (excluding benchmarks where Spot times out) are: 1.14 for random formulas, 1.12 for the real-world formulas of [BKS13], and 1.35 for the formulas of Acacia.

6   Conclusion

We have presented a simple, "Safraless", and asymptotically optimal translation from LTL and LDBA to deterministic parity automata. Furthermore, the translation is suitable for an on-the-fly implementation. The resulting automata are substantially smaller than those produced by the SPOT library for formulas obtained from synthesis specifications, and have comparable or smaller size for other benchmarks. In future work we want to investigate the performance of the translation as part of a synthesis toolchain.

References

[BHS+16] Blahoudek, F., Heizmann, M., Schewe, S., Strejček, J., Tsai, M.-H.: Complementing semi-deterministic Büchi automata. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 770–787. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49674-9_49

[BKS13] Blahoudek, F., Křetínský, M., Strejček, J.: Comparison of LTL to deterministic rabin automata translators. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 164–172. Springer, Heidelberg (2013). doi:10.1007/978-3-642-45221-5_12

[CY95] Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM **42**(4), 857–907 (1995)

[DLLF+16] Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, É., Xu, L.: Spot 2.0 — a framework for LTL and ω-automata manipulation. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 122–129. Springer, Heidelberg (2016). doi:10.1007/978-3-319-46520-3_8

[DWDMR08] Wulf, M., Doyen, L., Maquet, N., Raskin, J.-F.: Antichains: alternative algorithms for LTL satisfiability and model-checking. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 63–77. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78800-3_6

[EKRS17] Esparza, J., Křetínský, J., Raskin, J.-F., Sickert, S.: From LTL and limit-deterministic Büchi automata to deterministic parity automata. Technical report, https://arxiv.org/abs/1701.06103 (2017)

[Fin15] Finkbeiner, B.: Automata, games, and verification (2015). https://www.react.uni-saarland.de/teaching/automata-games-verification-15/downloads/notes.pdf

[FKVW15] Fogarty, S., Kupferman, O., Vardi, M.Y., Wilke, T.: Profile trees for Büchi word automata, with application to determinization. Inf. Comput. **245**, 136–151 (2015)

[JBB+16] Jacobs, S., Bloem, R., Brenguier, R., Khalimov, A., Klein, F., Könighofer, R., Kreber, J., Legg, A., Narodytska, N., Perez, G.A., Raskin, J.-F., Ryzhyk, L., Sankur, O., Seidl, M., Tentrup, L., Walker, A.: The 3rd reactive synthesis competition (SYNTCOMP 2016): benchmarks, participants & results. CoRR, abs/1609.00507 (2016)

[KR10] Kupferman, O., Rosenberg, A.: The blow-up in translating LTL to deterministic automata. In: Meyden, R., Smaus, J.-G. (eds.) MoChArt 2010. LNCS (LNAI), vol. 6572, pp. 85–94. Springer, Heidelberg (2011). doi:10.1007/978-3-642-20674-0_6

[KV01] Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Trans. Comput. Log. **2**(3), 408–429 (2001)

[KV15] Kini, D., Viswanathan, M.: Limit deterministic and probabilistic automata for LTL\GU. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 628–642. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46681-0_57

[Löd99] Löding, C.: Optimal bounds for transformations of ω-automata. In: Rangan, C.P., Raman, V., Ramanujam, R. (eds.) FSTTCS 1999. LNCS, vol. 1738, pp. 97–109. Springer, Heidelberg (1999). doi:10.1007/3-540-46691-6_8

[Pit07] Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. Logical Methods Comput. Sci. **3**(3:5), 1–21 (2007)

[Red12] Redziejowski, R.R.: An improved construction of deterministic omega-automaton using derivatives. Fundam. Inform. **119**(3–4), 393–406 (2012)

[Saf88] Safra, S.: On the complexity of omega-automata. In: FOCS, pp. 319–327 (1988)

[SEJK16] Sickert, S., Esparza, J., Jaax, S., Křetínský, J.: Limit-deterministic Büchi automata for linear temporal logic. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 312–332. Springer, Heidelberg (2016). doi:10.1007/978-3-319-41540-6_17

[ST03] Sebastiani, R., Tonetta, S.: "More deterministic" vs. "smaller" Büchi automata for efficient LTL model checking. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 126–140. Springer, Heidelberg (2003). doi:10.1007/978-3-540-39724-3_12

[Var85] Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: FOCS, pp. 327–338 (1985)

[VW86] Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: LICS, pp. 332–344 (1986)