International Journal on Software Tools for Technology Transfer
https://doi.org/10.1007/s10009-022-00663-1

GENERAL

Special Issue: TACAS 2017

From linear temporal logic and limit-deterministic Büchi automata to deterministic parity automata

Javier Esparza[1] · Jan Křetínský[1] · Jean-François Raskin[2] · Salomon Sickert[1,3]

Accepted: 7 June 2022

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 translations from LTL to LDBA to yield a double exponential, 'Safraless' LTL-to-DPA construction. We also report on an implementation and a comparison with other LTL-to-DPA translations on several sets of formulas from the literature.

Keywords Linear temporal logic · Deterministic parity automata · Synthesis

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 [4,39,40]. The problem faced by these authors was that fully nondeterministic Büchi automata (NBAs), which can capture all LTL-recognisable languages, cannot be used for probabilistic model checking, and deterministic Büchi automata (DBA), which could be used for probabilistic model checking, cannot capture all LTL-recognisable languages. 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 [32] 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 [2]. Kini and Viswanathan have given a single exponential translation of $\mathrm{LTL}_{\setminus \mathbf{GU}}$ to LDBA [18]. Finally, Sickert *et al.* describe in [9,34,36] two double exponential translations for full LTL that can also be applied to the quantitative case, and tend to behave better than Safra's construction in practice.

This work is partly supported by the ERC project *PaVeS* under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367, by the DFG projects *Verified Model Checkers* (317422601, 436811179) and *Group-By Objectives in Probabilistic Verification* (427755713), by the ARC project *Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond* (Fédération Wallonie-Bruxelles), the EOS project *Verifying Learning Artificial Intelligence Systems* (F.R.S.-FNRS & FWO), and the COST Action 16228 GAMENET (European Cooperation in Science and Technology).

✉ Jan Křetínský
jan.kretinsky@in.tum.de

Javier Esparza
esparza@in.tum.de

Jean-François Raskin
jraskin@ulb.ac.be

Salomon Sickert
sickert@in.tum.de

[1] Technische Universität München, München, Germany

[2] Université libre de Bruxelles, Bruxelles, Belgium

[3] The Hebrew University, Jerusalem, Israel

Published online: 23 July 2022

In this paper, we add to this trend by showing that LDBAs are also attractive for synthesis. The classical approach to the synthesis problem with LTL objectives[1] involves a translation of NBAs to DPAs with the help of the Safra-Piterman construction [30] or other recent determinisation constructions, such as [13,17,25]. While limit-determinism is not 'deterministic enough' for the synthesis problem, we introduce a conceptually simple and worst-case optimal translation LDBA→DPA.

The presented translation bears some similarities with that of [12] 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. Furthermore, readers familiar with [13,17] might notice that our construction tries to identify an (infinite) *left-path*, which is by definition accepting, in the *reduced split-tree*. If we restrict ourselves to LDBAs, identifying these becomes considerably simpler compared to the cited approaches. Hence our approach uses similar ideas, but is stream-lined and simpler.

Together with the translation LTL→LDBA of [9,34,36], 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. We solve this problem by showing that these LDBAs derived from LTL formulas possess semantic state annotations that can be used to reduce the amount of tracked information in the constructed DPA. We then prove that in this setting the concatenation of the two constructions remains double exponential.

With the availability of efficient translations from LTL formulas into DPAs several tools emerged following the classical approach to synthesis with LTL objectives. First, there is `ltlsynt` which is part of `Spot` [5] that uses a NBA→DPA translation. Second, there is `Strix` [27,28] that relies on the translation presented in this paper and which recently won all LTL tracks of the synthesis competition SyntComp [16]. Besides, the preserved semantic labelling of the states of the automata allows for heuristics guiding the exploration of the on-the-fly generated automaton [27], but also for efficient deployment of learning-based algorithms and lifelong learning paradigms in LTL synthesis [19]. Such efforts have a great impact on the practical performance of solutions to this 2-EXPTIME-complete problem. For a detailed description of the exact implementation details of `Strix` we refer the reader to [27].

[1] See [3] for an in-depth description of the classical and recent approaches to LTL synthesis.

In the third and final part, we report on an experimental evaluation of our LTL→LDBA→DPA construction, and compare it with other constructions that translate LTL to DPAs.

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 this translation with a translation from LTL to LDBAs 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.

Editorial Note. This is an extended journal version of our previously published conference paper [8], including full proofs, more examples, and an extensive evaluation on classical and new, parametrised benchmarks.

2 Preliminaries

Büchi automata. A (nondeterministic) 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[2]. A is *deterministic* if for all $q \in Q$, for all $\sigma \in \Sigma$, there exists a unique $q' \in Q$ such that $(q, \sigma, q') \in \delta$ or there exists no such state. Given $S \subseteq Q$ and $\sigma \in \Sigma$, let $\mathsf{post}^{\sigma}_{\delta}(S) = \{q' \mid \exists q \in S \cdot (q, \sigma, q') \in \delta\}$. Further, we use $q \to^{\sigma} p$ as a shorthand for $(q, \sigma, p) \in \delta$ if δ is clear from the context.

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] 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.

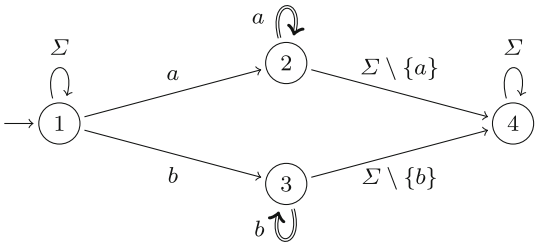

Fig. 1 An LDBA for the LTL language $\mathbf{FG}a \vee \mathbf{FG}b$. The behaviour of A is deterministic within the subset of states $Q_d = \{2, 3, 4\}$ which is a trap, the set of accepting transitions are depicted in bold face and they are defined only between states of Q_d. We simplify figure, by using the alphabet $Ap = a, b$ instead of 2^{Ap}

2. $\forall q \in Q_d \cdot \forall \sigma \in \Sigma \cdot \forall q_1, q_2 \in Q \cdot (q, \sigma, q_1) \in \delta \wedge (q, \sigma, q_2) \in \delta \rightarrow q_1 = q_2$, i.e. the transition relation δ is deterministic within Q_d;
3. $\forall q \in Q_d \cdot \forall \sigma \in \Sigma \cdot \forall q' \in Q \cdot (q, \sigma, q') \in \delta \rightarrow q' \in Q_d$, i.e. Q_d is a trap (when Q_d is entered it is never left).

Without loss of generality, 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 $\mathsf{L}(A) = \mathcal{L}$ [4]. 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 recognised by a deterministic Büchi automaton.

Parity automata. A deterministic 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 a function assigning an integer in $\{1, 2, \ldots, d\}$, called a *colour*, to each transition in the automaton. Colours are naturally ordered by the order on integers.

Given a run ρ over a word w, the infinite sequence of colours traversed by the run ρ is noted $p(\rho)$ and is equal to $p(\rho(0), w(0), \rho(1)) \ldots\ p(\rho(n), w(n), \rho(n+1)) \ldots$. A run ρ is *accepting* if the minimal colour 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. [30].

Linear Temporal Logic. We introduce linear temporal logic (LTL) as most authors with the following reduced syntax:

$$\varphi := \mathbf{tt} \mid a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \quad \text{with } a \in Ap$$

Let w be a word over the alphabet 2^{Ap} and let φ be a formula. Let $w_i = w(i)w(i+1) \ldots$ denote the suffix of w at position i. The satisfaction relation $w \models \varphi$ is inductively defined as follows:

$$\begin{array}{ll} w \models \mathbf{tt} & \\ w \models a & \text{iff } a \in w(0) \\ w \models \neg\varphi & \text{iff } w \not\models \varphi \\ w \models \mathbf{X}\varphi & \text{iff } w_1 \models \varphi \\ w \models \varphi \mathbf{U} \psi & \text{iff } \exists k \cdot w_k \models \psi \text{ and } \forall j < k \cdot w_j \models \varphi \end{array}$$

We denote by $\mathsf{L}(\varphi) := \{w \in (2^{Ap})^\omega \mid w \models \varphi\}$ the language of φ. Left-out, but often used LTL operators are then added as abbreviations. $\mathbf{F}\varphi := \mathbf{tt}\mathbf{U}\varphi$ (eventually) and $\mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$.

3 From LDBA to DPA

3.1 Run DAGs and their colouring

Run DAG. A nondeterministic automaton A may have several (even an infinite number of) runs on a given ω-word w. As in [23], 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 $i \geq 1$,
  - $V_i = \{(q, i) \mid \exists (q', i-1) \in V_{i-1} \cdot q' \rightarrow^{w(i)} q\}$;
- $E = \{((q, i), (q', i+1)) \in V_i \times V_{i+1} \mid q \rightarrow^{w(i)} q'\}$.

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 *infinite* 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 infinite 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 $\mathsf{Ord} : Q \rightarrow \{1, 2, \ldots, |Q_d|, +\infty\}$ 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 $\rho(0..n) = v_0 v_1 \dots v_n$ and $\rho'(0..n) = v'_0 v'_1 \dots v'_n$ 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 $\{1, 2, \dots, |Q_d|\}$ 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
  - either there does *not* exists $v' \in V_{i+1}^d : (v,v') \in E$, i.e. v has no successor in V_{i+1}^d meaning that the sequence of states monitored so far aborts and does not lead to an infinite run;
  - or 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 (infinite) 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 stabilise.

Assigning colours. The set of colours that are used for colouring the levels of the run DAG G_w is $\{1, 2, \dots, 2 \cdot |Q_d| + 1\}$. We associate a colour 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 colour 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 colour 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 colour is defined as the minimal colour 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 colour is $2 \cdot |Q_q| + 1$.

The intuition behind this colouring is as follows: the colouring tracks (potentially infinite) runs in Q_d, as $\alpha \subseteq Q_d \times \Sigma \times Q_d$, and tries to produce an even colour 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 colour 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 or they may *abort*. When this happens, this is considered as a *negative event* because the even colours that have been emitted by the run that merges with the smaller run or aborts should not be taken into account anymore. As a consequence an odd colour is emitted in order to cancel all the (good) even colours that were generated by the run that merges or aborts. In that case the odd colour is function of the smallest index of a run vertex in V_i whose run merges or aborts. 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 colour is determined by the minimum of the two colours 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 or abort, the largest odd colour is emitted as indicated by case 4 above.

According to this intuition, we define the *colour summary* of the run DAG G_w as the minimal colour that appears infinitely often along the transitions between its levels. Because of the deterministic behaviour 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

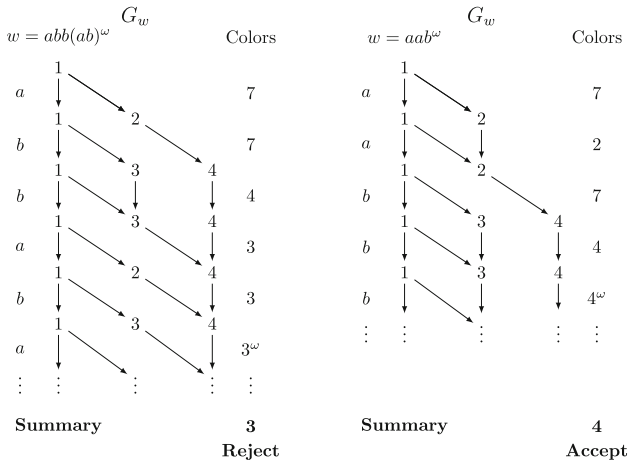

Fig. 2 The run DAGs automaton of Fig. 1 on the word $w = (ab)^\omega$ given on the left, and on the word $w = aab^\omega$ given on the right, together with their colourings

colouring, we know that, on word accepted by A, the smallest accepting run will eventually generate infinitely many (good) even colours that are never trumped by smaller odd colours.

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 colouring of the run DAG follows the colouring rules defined above. Between level 0 and level 1, the colour 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 colour 7 is also emitted from level 1 to level 2 for the same reason. The colour 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 colour 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 colours emitted by the run that goes from 3 to 4, colour 3 is emitted. It cancels the even colour 4 emitted before by this run. Afterwards, colours 3 is emitted forever. The colour 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 colouring of the run DAG follows the colouring rules defined above. Between levels 0 and 1, colour 7 is emitted because no accepting edge is crossed. To the next level, we see the accepting edge $(2, a, 2)$ and colour $2 \cdot 1 = 2$ 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 colour is forever equal to 4. The colour 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 colour summary (the minimal colour that appears infinitely often) can be used to identify run DAGs that contain accepting runs.

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

Proof ($\Rightarrow$): Assume that the colour summary of G_w is even and equal to c. Then it must be the case that there exists a level $i \geq 0$ such the colour after level i is always larger than or equal to c, and infinitely many times equal to c. W.l.o.g. assume that in level i, there exists a vertex $v = (q, i) \in \mathsf{Acc}(V_i^d)$ and $c = 2 \cdot \mathsf{Ind}(v)$. Take the smallest run prefix that ends up in v, this run prefix will never merge with a smaller run prefix, and all smaller run prefixes that are active in level i will not merge or abort, as otherwise, there would exist a position $j \geq i$ where the index of the run that passes by (q, i) would decrease and this would contradict the fact that for all $j \geq i$, all the colours that are emitted are larger than or equal to c. Let us now consider the suffix of the run that pass by $v = (q, i)$. As the even colour c is emitted infinitely many times after level i, we know that this run suffix crosses infinitely many times α. So this run is accepting and this is the smallest such run.

($\Leftarrow$): (Step 1): Now, let us consider the other direction. Assume that there exists an accepting run of A on a word w. We first establish the existence of a run ρ which is accepting and for which there exists a position $k \geq 0$ from which ρ does not merge with any smaller run, and all smaller runs are non accepting. We identify ρ and k as follows. Among the accepting runs, we select one that enters first in the set of states Q_d say at level $i \geq 0$. They can be several of them, but we take one that enters Q_d via a state q of minimal index for Ord. Let V_i^d be the active states at level i that are in Q_d. The way we have chosen q make sure that all the states in V_i^d with a smaller index than q are the origin of non accepting runs and clearly as ρ is accepting it cannot merge with one of those smaller runs. Now, some of those smaller runs may merge or abort in the future, and each time they merge or abort, the index of ρ will decrease. But this will happen a number of times which is bounded by Q_d.

(Step 2): Let k be the position when the last merge or abort of a smaller run prefix happens.

(Step 3): Let us now show that the existence of ρ and this position k allow us to prove that the colour summary is even. After position k, there are only odd colours with values larger than or equal to $2 \cdot \mathsf{Ind}(\rho(k)) + 1$ because we know that nor ρ neither smaller runs merge or abort in the future. Also as ρ is accepting, there will be an infinite number of positions $l \geq k$ where the even colour is equal to $2 \cdot \mathsf{Ind}(\rho(k)))$, and

only finitely many positions after k may have an even colour which is less than this value as all runs that are smaller than ρ are not accepting. So the summary colour is even and equal to $2 \cdot \mathsf{Ind}(\rho(k)))$. □

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 colouring 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 $\sigma \in \Sigma$, and let us assume that there is a state $q \in s_1 \cup t_1$ and a state $q' \in Q$ such that $(q, \sigma, q') \in \delta$ (otherwise δ^B is not defined in $(s_1, (t_1, <_1))$ for σ). Then $\delta^B((s_1, (t_1, <_1)), \sigma) = (s_2, (t_2, <_2))$ 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: $\forall q_1, q_2 \in t_2$: $q_1 <_2 q_2$ 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.

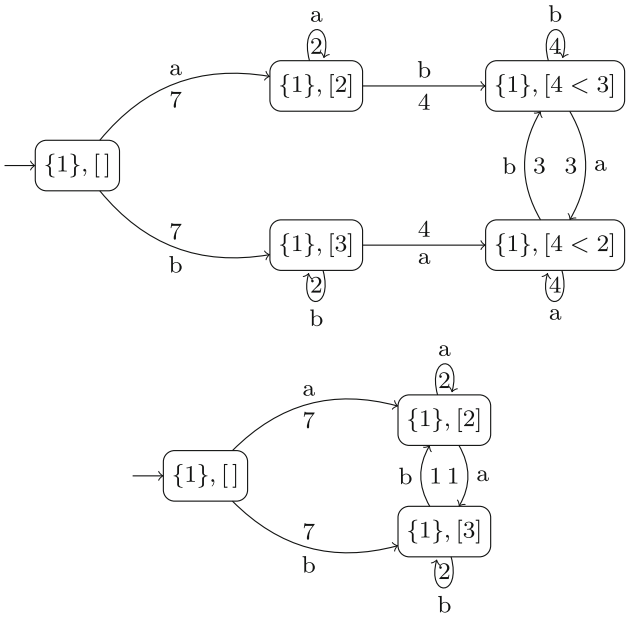

Fig. 3 Upper: DPA that accepts the LTL language $\mathbf{FG}a \vee \mathbf{FG}b$, edges are decorated with a natural number that specifies its colour. Lower: a reduced DPA

Colouring. To define the colouring 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 or that abort because they have no δ-successor for σ. Those are defined as follows:

$$\mathsf{Dec}(t_1) = \left\{ q_1 \in t_1 \mid \begin{array}{l} \mathsf{Ind}_{(t_2,<_2)}(\delta(q_1, \sigma)) < \mathsf{Ind}_{(t_1,<_1)}(q_1) \\ \vee \neg\exists(q_1, \sigma, q) \in \delta \end{array} \right\}.$$

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 colour 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 colour 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 colour 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 colour is defined as the minimal colour 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 colour is $2 \cdot |Q_d| + 1$.

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 recognises 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 an ordered subset of Q_d of the original NBA. As an illustration of the definitions above, let us explain the colour 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 colour 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 colour 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 colour 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 colours that occur along G_w is equal to the sequence of colours 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

3.3.1 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):

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

Thus the obtained DPA has $\mathcal{O}(2^n \cdot 2^{n \log n}) \subseteq 2^{\mathcal{O}(n \log n)}$ states and $2n_d + 1 \in \mathcal{O}(n)$ colours.

3.3.2 Lower bound

We obtain a matching lower bound by strengthening Theorem 8 from [24]:

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 recognised by a limit-deterministic Büchi automaton with $3n + 2$ states but cannot be recognised by a deterministic Parity automaton with less than $n!$ states.*

Proof The proof of Theorem 8 from [24] constructs a nondeterministic 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 [24]. □

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 DPA in $2^{2^{\mathcal{O}(n)}}$

In [34,36], we present two different LTL→LDBA translations. Given a formula φ of size n, both translations produce an asymptotically optimal LDBA with $2^{2^{\mathcal{O}(n)}}$ states. The straightforward composition of these translations with the single exponential LDBA→DPA translation of the previous section is only guaranteed to be triple exponential, while the Safra–Piterman and Muller–Schupp constructions produce DPAs of at most doubly exponential size, if applied to NBAs constructed from LTL formulas.

In this section, we describe two modifications of our simple approach relying on additional semantic information that yield DPAs with $2^{2^{\mathcal{O}(n)}}$ states. The approach taken by both modifications is the following: We can view the second component of the states produced by our construction as a sequence of states of the LDBA, ordered by their indices. Since there are $2^{2^{\mathcal{O}(n)}}$ states in the LDBA for an LTL formula of length n, the number of such sequences is:

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

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

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

Both modifications prune these state sequences and guarantee that their length stay below a suitable threshold such that the resulting DPAs are asymptotically optimal.

4.1 Pruning by language decomposition

We introduce the main ideas of this approach using the LDBA[3] depicted in Fig. 4 and the corresponding DPA

[3] Observe that this language can also be recognised by a DBA. However, this particular LDBA illustrates the main idea of construction.

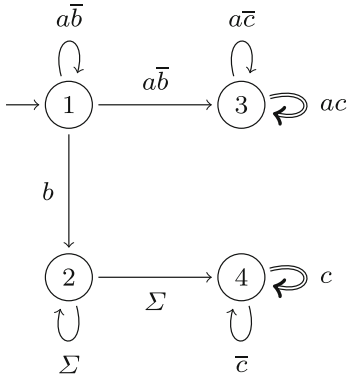

Fig. 4 An LDBA A for the LTL language $a\mathbf{W}b \wedge \mathbf{GF}c$.[3] The behaviour of A is deterministic within the subset of states $Q_d = \{3, 4\}$ which is a trap and contains all accepting transitions which are depicted in bold face

depicted in Fig. 5 (upper part) obtained by the construction from the previous section. First, let us examine the LDBA: the state q_4 accepts a superset of the language accepted by q_3 and state q_2 allows to 'restart' failed runs in q_4. Second, let us examine the DPA: the state $\langle\{2\}, [3 < 4]\rangle$ encodes that in the corresponding run DAG the run in q_3 has entered first Q_d before the run in q_4. One also immediately sees that the states $\langle\{2\}, [3]\rangle$ and $\langle\{2\}, [3 < 4]\rangle$ are bi-similar and that they can be collapsed to a single state (lower part). However, inspecting this example closer we can find a different explanation for this phenomenon. One sees that the states q_3 and q_4 in the original LDBA accept if c appears infinitely often and only differ in the treatment of a. In fact this can be captured by two classic notions about languages:

- A language $S \subseteq \Sigma^\omega$ is a *safety* language if there exists a set of *bad* prefixes $B \subseteq \Sigma^*$ such that $S = \Sigma^\omega - B\Sigma^\omega$. Thus for all words outside the language there exists a finite witness that the word does not belong to the language. We then denote the set of all safety languages by $\mathcal{S}$.
- A language $C \subseteq \Sigma^\omega$ is a *suffix-closed* language if $C \subseteq \Sigma C$. Thus all suffixes of a word in the language are also in the language. We then denote the set of all suffix-closed languages by $\mathcal{C}$.

The languages of q_3 and q_4 are in fact an intersection of languages from $\mathcal{S}$ and $\mathcal{C}$. To be more concrete, we have $\mathsf{L}(q_3) = C \cap S_1$ and $\mathsf{L}(q_4) = C \cap S_2$ with $C = \mathsf{L}(\mathbf{GF}c)$, $S_1 = \mathsf{L}(\mathbf{G}a)$, $S_2 = \Sigma^\omega$. We now make use of this to remove nodes from the run-DAG and thus also explain the removal of $\langle\{2\}, [3 < 4]\rangle$.

Assume we have the situation $V_i = \{q_2, q_3\}$, $V_i^d = \{q_3\}$, and $V_{i+1}^d = \{q_3, q_4\}$. We argue that we can keep only q_3, redefine $V_{i+1}^d := \{q_3\}$, and still capture all relevant information to decide acceptance. We focus on the difficult case where the subtree of $q_3 \in V_{i+1}^d$ is rejecting and the subtree of $q_4 \in V_{i+1}^d$

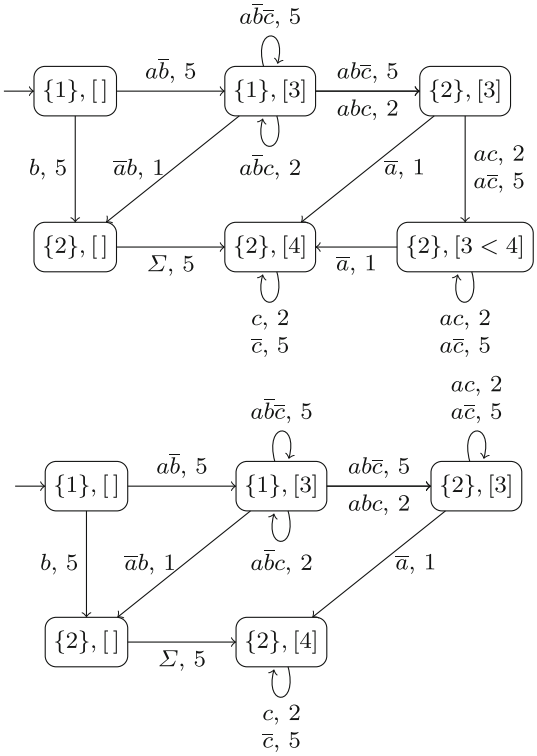

Fig. 5 Upper: DPA constructed for the LDBA from Fig. 4, edges are decorated with a natural number that specifies its colour. Lower: A reduced DPA obtained through the construction relying on Proposition 1

is accepting. Then since q_4 is accepting the suffix w_{i+1} is in C and thus w_{i+1} cannot be in S_1, which is the safety condition for q_3. Since S_1 is a safety language, we will detect this after a finite prefix and can discard that particular branch of the run-DAG. Thus for some $j > i$ we have $V_j^d = \{\}$. Since we have $V_j = \{q_2\}$ due the self-loop on q_2, we get $V_{j+1}^d = \{q_4\}$ and this subtree is going to be accepting, since $w_i \in C$ and thus also $w_j \in C$ due to the suffix-closure of C.

Let us now generalise these insights: We call an LDBA *decomposable* if $\delta \cap (\overline{Q_d} \times \Sigma \times \overline{Q_d})$ is deterministic, and there exists a partition of the states Q_d into sets Q_d^1, Q_d^2, $\ldots Q_d^n$ such that for each Q_d^i the following holds:

1. $\exists C \in \mathcal{C} \cdot \forall q \in Q_d^i \cdot \exists S \in \mathcal{S} \cdot \mathsf{L}(q) = S \cap C$, i.e., all states in the component Q_d^i can be represented by an intersection of a safety language and a suffix-closed language,
2. $\forall q \in Q_d^i \cdot \forall \sigma \in \Sigma \cdot \forall q' \in Q \cdot (q, \sigma, q') \in \delta \rightarrow q' \in Q_d^i$, i.e. Q_d^i is a trap (when Q_d^i is entered it is never left),
3. if $q \rightarrow^\sigma p \rightarrow^{\sigma'} r$ for states $q \in \overline{Q_d}$, $p, r \in Q_d^i$ and letters $\sigma, \sigma' \in \Sigma$, then there exists $p' \in \overline{Q_d}$ and $r' \in Q_d^i$ such that $q \rightarrow^\sigma p' \rightarrow^{\sigma'} r'$ and $\mathsf{L}(r) \subseteq \mathsf{L}(r')$, i.e. moving to the partition Q_d^i can be delayed, and
4. $\forall q \in \overline{Q_d} \cdot \forall \sigma \in \Sigma \cdot |\delta(q, \sigma) \cap Q_d^i| \leq 1$, i.e. for each state $q \in \overline{Q_d}$ and letter σ we have at most one transition to Q_d^i.

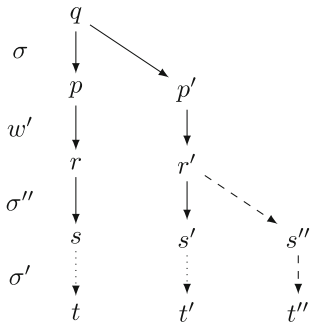

Fig. 6 Structure of the induction step in Lemma 3

In fact, we can obtain by repetitive application of assumption (3) a generalisation to arbitrary finite words:

Lemma 3 *Assume* $q \to^{\sigma} p \to^{w} r \to^{\sigma'} s$ *for states* $q \in \overline{Q_d}$, $p, r, s \in Q_d^i$ *and letters* σ, σ' *and (finite) word* $w \in \Sigma^*$, *then there exists* $p', r' \in \overline{Q_d}$ *and* $s' \in Q_d^i$ *such that* $q \to^{\sigma} p' \to^{w} r' \to^{\sigma'} s'$ *and* $\mathsf{L}(s) \subseteq \mathsf{L}(s')$.

Proof We proceed by induction on w. In the case $w = \epsilon$ we can immediately apply assumption (3).

Case $w = w'\sigma''$: Consider Fig. 6. In terms of this picture we need to prove that there exists some $t'' \in Q_d^i$ such that $\mathsf{L}(t) \subseteq \mathsf{L}(t'')$. We obtain the first part (the solid lines) by applying the induction hypothesis and have $\mathsf{L}(s) \subseteq \mathsf{L}(s')$ such that $s, s' \in Q_d^i$. Since the transition relation within Q_d^i is deterministic we also obtain $\mathsf{L}(t) \subseteq \mathsf{L}(t')$ (the dotted lines). Now we apply, assumption (3) on r', s', and t' (the dashed lines) to obtain s'' and t'' such that $\mathsf{L}(t') \subseteq \mathsf{L}(t'')$. Then by transitivity we get $\mathsf{L}(t) \subseteq \mathsf{L}(t'')$. □

We claim that for each block of the partition Q_d^i at most one state needs to be tracked by a run DAG in order to decide acceptance. Without loss of generality let us assume that LDBAs only contain states that are reachable from the initial state and that can reach an accepting transition. Thus for any state $q \in Q$ we have $\mathsf{L}(q) \neq \emptyset$. Given a decomposable LDBA with a set of states Q and a suitable partition Q_d^1, Q_d^2, $\ldots Q_d^n$, we define the *reduced run DAG* G_w^*. 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:

- $V = \bigcup_{i \in \mathbb{N}} V_i$ and $V_i = \bigcup_{j=0}^{n} V_{i,j}$, where $V_{i,0}$ contains the nodes representing runs that are at level i in $\overline{Q_d}$ and $V_{i,j}$ contains the nodes that correspond to Q_d^j. Formally, the sets $V_{i,j}$ are defined inductively as:

$$
\begin{aligned}
V_{0,0} &= \{(q_0, 0)\} \\
V_{0,j} &= \emptyset \\
V_{i,0} &= \mathsf{post}_{\delta}^{w(i-1)}(V_{i-1,0}) \cap (\overline{Q_d} \times \{i\}) \\
V_{i,j} &= \begin{cases} \mathsf{post}_{\delta}^{w(i-1)}(V_{i-1,0}) \cap (Q_d^j \times \{i\}) & \text{if } V_{i-1,j} = \emptyset \\ \mathsf{post}_{\delta}^{w(i-1)}(V_{i-1,j}) & \text{otherwise.} \end{cases}
\end{aligned}
$$

  for all $i \geq 1$ and all $1 \leq j \leq n$ and where we use

$$\mathsf{post}_{\delta}^{\sigma}(V_{i,j}) = \{q \in Q \mid \exists (q', i) \in V_{i,j} \cdot (q', \sigma, q) \in \delta\}$$

  to denote the successors of a level in the underlying automaton.
- $E = \{((q, i), (q', i+1)) \in V_i \times V_{i+1} \mid q \to^{w(i)} q'\}$.

Let us now reconsider the LDBA from Fig. 4. This LDBA is decomposable with $Q_d = Q_d^1$. Property (1) follows from our previous analysis and (2) is a direct consequence of the LDBA definition, since we only have single partition, and (4) follows from the fact that we have at most one transition from q_1 and q_2 to Q_d under each letter. For (3) observe that $\mathsf{L}(q_3) \subseteq \mathsf{L}(q_4)$ and we have the 'restarting' loop in q_2. Let us now see why this pruning is correct:

Proposition 1 *There is an accepting run in the reduced run DAG* G_w^* *if and only if there is an accepting run in the run DAG* G_w.

Proof ($\Rightarrow$) Observe that G_w^* is a subgraph of G_w, since we obtain G_w^* from G_w by removing nodes and edges. Thus every accepting run on G_w^* is also an accepting run on G_w.

($\Leftarrow$) Assume that $G_w = (V, E)$ has an accepting run. Then an accepting run eventually transitions from some $q \in \overline{Q_d}$ by reading the letter $w(i-1)$ to some $p \in Q_d^j$. Then $p \in V_i^d$ and the (deterministic) run starting in (p, i) is accepting. Let us now see what happens in $G_w^* = (V', E')$. We proceed by a case distinction.

Assume $V'_{i,j} = \{p\}$. Then by definition of G_w^* we also have a (deterministic) run starting in (p, i) that is identical to the one in G_w starting in (p, i) which is accepting.

Assume $V'_{i,j} = \emptyset$ and let $r = \delta(p, w(i))$. Then by assumption (3) and (4) there exists unique $p' \in \overline{Q_d}$ and $r' \in Q_d^j$ such that $q \to^{w(i-1)} p' \to^{w(i)} r'$ and $\mathsf{L}(r) \subseteq \mathsf{L}(r')$. Since the (deterministic) run $(r, i+1)$ (in G_w) is accepting, the (deterministic) run starting $(r', i+1)$ (in G_w^*) is also accepting.

It remains to consider the case where $V'_{i,j}$ is neither empty nor simply $\{p\}$. By the definition of G_w^* there exists a unique state $p' \in V'_{j,i}$. If $w_i \in \mathsf{L}(p')$, then we are also done since then G_w^* has then an accepting run. Thus assume $w_i \notin \mathsf{L}(p')$. From $w_i \in \mathsf{L}(p)$ we derive that $w_i \in C$ for the suffix-closed language C associated with Q_d^j. This follows from assumption (1). Moreover, assumption (1) tells us that $w_i \notin S$ for all $S \in \mathcal{S}$ with $\mathsf{L}(q') = C \cap S$. Finally, since the LDBA does not contain states q with $\mathsf{L}(q) = \emptyset$, there must be a level $i' > i$ such that $V'_{i',j}$ is empty. We then proceed analogous to the case $V'_{i,j} = \emptyset$, but make use of Lemma 3 to bridge the longer distance. To be more precise, let $q \to^{w(i-1)} p \to^{w(i)} \cdots \to^{w(i'-2)} r \to^{w(i'-1)} s$ be the sequence of states in the original run-DAG. We then apply

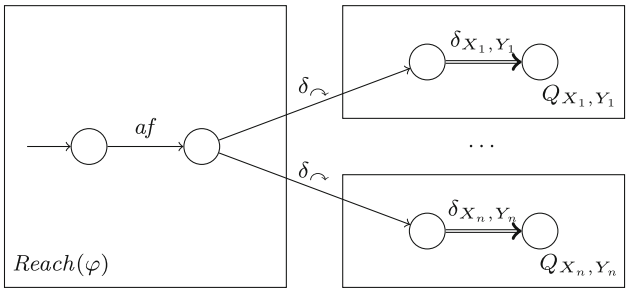

Fig. 7 Schematic overview of an LDBA obtained in [34, Theorem 6.2] for a formula φ. $\overline{Q_d}$ is on the left and Q_d is on the right hand-side

Lemma 3 to obtain $s' \in Q_d^j$ with $\mathsf{L}(s) \subseteq \mathsf{L}(s')$ and that is in $V'_{i'+1,j}$. Due to the language inclusion, this run is then accepting. □

We now can apply the construction from Sect. 3.2 to obtain a DPA tracking the reduced run DAG G_w^*. Assume that the LDBA has m states and n partitions. Then each V_i has at most cardinality $n+1$ and thus the resulting DPA has at most $(m+1)^{n+1}$ states.

A suitable LTL → LDBA translation. We now show that decomposable LDBAs exist and these are in fact produced by the recent LTL→LDBA translation defined in [34][Theorem 6.2].

Proposition 2 *For all LTL formulas φ, the procedure of [34] [Theorem 6.2] produces a LDBA which is decomposable and has at most 2^n partitions, where n is the the length of φ.*

Proof Let us first sketch the structure of the resulting LDBA. A schema of the structure can be found Fig. 7. The states of the LDBA are a disjoint union of the set $Reach(\varphi)$, forming the states for the initial component, and Q_{X_i,Y_i}, forming the states for the accepting component. Further, the latter is parametrised by sets X_i and Y_i which depend on the formula φ. Within the initial component, we have a deterministic transition relation, named af, and states that can be identified by LTL formulas. The accepting component is constructed as follows: for fixed sets X and Y, let $A_{X,Y} = (Q_{X,Y}, q_{0,X,Y}, \Sigma, \delta_{X,Y}, \alpha)$ be the intersection of the following deterministic Büchi automata (DBA):

- $A^1_{\varphi,X}$ accepts the language of a syntactic safe formula obtained from φ and X.
- $A^2_{X,Y}$ accepts the language of $\bigwedge_i \mathbf{GF}\psi_i$, where ψ_i is derived from X and Y.
- $A^3_{X,Y}$ accepts the language of $\bigwedge_j \mathbf{G}\psi_j$, where ψ_j is a syntactic safe formula derived from X and Y.

The overall transition relation δ for the LDBA A is the union of af, $\delta_{X,Y}$, and the yet-to-be-defined $\delta_{\curvearrowright}$. $\delta_{\curvearrowright}$ connects the initial component with the accepting component. It contains exactly one edge for each state in $Reach(\varphi)$ to a state in $Q_{X,Y}$.

We now claim that the partition $\overline{Q_d} = Reach(\varphi)$, $Q_d^1 = Q_{X_1,Y_1}$, $Q_d^2 = Q_{X_2,Y_2}$, ..., $Q_d^m = Q_{X_m,Y_m}$ is a suitable partition to show that the LDBA A is decomposable. For this observe, that af, $\delta_{X,Y}$ are deterministic transition relations. Lastly, we need to verify that for each $Q_{X,Y}$ the assumptions (1-4) hold:

1. By construction, $A_{X,Y}$ is derived by an intersection, every state in $q \in Q_{X,Y}$ recognises the intersection $\mathsf{L}(\bigwedge_i \mathbf{GF}\psi_i) \in \mathcal{C}$ and a safety language.
2. By construction Q is a disjoint union and no transitions leaving $Q_{X,Y}$ have been added, thus $Q_{X,Y}$ is a trap.
3. This assumption is proven by the technical [34][Lemma 4.20] relating af and $\cdot[\cdot]_v$, which is the essential component of $\delta_{\curvearrowright}$.
4. Finally, it can be easily verified by looking at the definition $\delta_{\curvearrowright}$ in [34][Theorem 6.2], that there exists at most one transition from the initial component to each of the partitions.

Lastly we claimed that there are at most 2^n, where n is the size of the formula, partitions. For this observe that X_i and Y_j are according to [34][Theorem 6.2] subsets of two disjoint sets containing only subformulas of φ and thus there exist at most 2^n possible choices for X_i and Y_j. □

4.2 Pruning by language subsumption

Fix an LDBA with a 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 $\ell > k$).
- Remove any vertices (if any) that are no longer reachable from vertices of V_1.

We define the colour summary of G^*_w in exactly the same way as the colour summary of G_w. The DAG G^*_w satisfies the following crucial property:

Proposition 3 *The colour summary of the run DAG G^*_w is even if and only if there is an accepting run in G_w.*

Proof "$\Rightarrow$": The 'only-if' direction can be proven as in Theorem 1 verbatim, only replacing G_w by G^*_w. The reason why the argumentation is still correct, is that the discussed "smallest run prefix that ends up in v" (now in G^*_w) is actually a real run prefix (in G_w) since it never secondarily merged. Indeed, runs only merge into smaller ones.

"$\Leftarrow$": (Step 1): For the 'if' direction, we first use the proof Theorem 1, Step 1, verbatim, obtaining the smallest accepting run in G_w.

Additionally, we prove that this (smallest) constructed run ρ is actually a run in G^*_w. For a contradiction, assume that this is not the case and $\rho = \rho_1(v_k, i)\rho_2$ where (v_k, i) is the first vertex on ρ that secondarily merged. Then there is $(v_j, i) \in V_i \cap (Q_d \times \{i\})$ with $v_j \sqsubset v_k$ and $\mathsf{L}(v_j)$ contains the label of the run $(v_k, i)\rho_2$, accepted by some run $(v_j, i)\rho'_2$ in G_w. Since $(v_j, i) \sqsubset_i (v_k, i)$, we also have a run prefix $\rho'_1(v_j, i) \sqsubset \rho_1(v_k, i)$, and thus an accepting run $\rho'_1(v_j, i)\rho'_2$ in G_w such that $\rho'_1(v_j, i)\rho'_2 \sqsubset \rho_1(v_k, i)\rho_2 = \rho$, a contradiction with minimality of ρ.

(Step 2): Let k be the position when the last merge of a smaller run prefix happens in G^*_w (not G_w).

(Step 3): We use the proof Theorem 1, Step 3, verbatim, proving the colour summary is even. □

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 colour 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 colour 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 3: $\mathcal{A}$ accepts w iff G_w contains an accepting run iff the colour 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 parametrised 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 generalisation.

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 recognises a union of some base languages. (Note that every automaton has a base of at most linear size.) Whenever we have states v_j recognising $\bigcup_{i \in I_j} L_i$ 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 4 *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 (nonequivalent) 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. □

4.3 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 [36]. 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 "Appendix A".

Proposition 4 *For every LTL formula φ, every state s of the LDBA of [36] 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 [36] 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 $2^{\mathcal{O}(|\varphi|)} \cdot 2^{\mathcal{O}(|\varphi|)} = 2^{\mathcal{O}(|\varphi|)}$ □

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 4. Therefore, by Corollary 2, the size of the DPA is doubly exponential, in fact $2^{(2^{\mathcal{O}(|\varphi|)})^2} = 2^{2^{\mathcal{O}(|\varphi|)}}$ □

This matches the lower bound $2^{2^{\Omega(n)}}$ by [22] as well as the upper bound by the Safra-Piterman approach. Finally, note that while the breakpoint constructions in [36] is analogous to Safra's vertical merging, the merging introduced here is analogous to Safra's horizontal merging.

4.4 Comparing the two pruning methods

We presented two different pruning techniques to achieve an asymptotically optimal LTL→DPA translation. The question is how to they compare on conceptual level, is one stronger than the other? No, in fact they are incomparable. Consider Fig. 5. Here, the first construction removes the ranking $[3 < 4]$ but this cannot be achieved by the second construction. On the other hand, is is clear that language-based pruning technique can be applied to any LDBA (by using a language inclusions checks) and thus is applicable to larger set of LDBAs.

5 Experimental evaluation

We showed that our determinisation construction, which is considerably simpler compared to other constructions, can be combined with semantic pruning of tracked states. This then yields an asymptotical optimal construction. This simplicity is achieved by a detour over LDBAs and one would expect that this incurs inefficiencies in practice. In this section, we provide experimental evidence that this *not* the case.

5.1 Method

Metric. We compare the size (number of states, number of colours) of produced automata since this is a good indicator of the size of the arena in the automata-theoretic approach to synthesis. In contrast, we do not include any resource consumption analysis, i.e. measurements of computation time or allocated memory. Not only are these values highly dependent on implementation details but, foremost, as shown by [27] on-the-fly computation of the state-space and additional compositional constructions are highly relevant for the overall computation time in the context of synthesis. For each input formula φ_i, we compare sizes a_i and b_i achieved by different methods and are interested in the achieved improvement factor, which is their ratio a_i/b_i. In order to aggregate the factors into an average one, given that the involved numbers can be hugely different, it is appropriate to use the *geometric* average of the ratios:

$$\sqrt[n]{\prod_{i=1}^{n} \frac{a_i}{b_i}} = \frac{\sqrt[n]{\prod_{i=1}^{n} a_i}}{\sqrt[n]{\prod_{i=1}^{n} b_i}}$$

Consequently, for each approach and a set of inputs, we display the the geometric average of the sizes since, for each pair of approaches, the respective ratio immediately yields the desired comparison.

Competing Translations. We compare seven configurations from three different groups of translations that yield DPAs (with the acceptance condition defined on transitions):

– via NBAs:

N_1 `ltl2tgba`[4] ([5], `2.8.5`): The tool implements a *portfolio* approach for constructing small automata, in our configuration DPAs. If the formula is not covered by one of the specialised constructions, a version of the Safra determinisation procedure [31] with several optimisations is used. Thus one can argue that it is the state-of-the-art portfolio translator.

N_2 `nbadet`[5] ([26][6]): The tool implements the construction presented [25] with additional optimisations [26]. We use `ltl2tgba -B` to translate LTL formulas into NBAs with the acceptance condition defined on states, since in the tested version of the tool acceptance conditions on transitions are not supported. Note that such a detour causes a blowup in the intermediate NBA.

– via DRAs (deterministic Rabin automata):

D_1 `ltl2dgra (asymmetric)`, `dgra2dra`, `dra 2dpa`[7]: This approach uses the direct translation to deterministic generalised Rabin automata (DGRA) that has been described in [7] and revised and corrected in [10]. This configuration uses all available optimisations, including the usual reduction rules for Rabin pairs, e.g. [10]. This approach treats the least- (**F**, **U**, **M**) and greatest-fixed-point operators (**G**, **W**, **R**) 'asymmetrically', and has only a proven triple exponential upper bound. We combine it with the IAR (index-appearance record) construction as improved in [21].

D_2 `ltl2dra (symmetric)`, `dra2dpa`[7]: This approach uses the direct translation to DRAs based on the 'Master-Theorem' [9,34] and uses optimisations described in detail in [34]. As in the previous case, we combine it with the IAR construction of [21].

– via LDBAs:

LD_1 `ltl2dpa (asymmetric)`[7]: This translation combines the construction presented here with the 'asymmetric' translation LTL→LDBA presented in [36], which treats least- (**F**, **U**, **M**) and greatest-fixed-point operators (**G**, **W**, **R**) differently.

LD_2 `ltl2dpa (symmetric)`[7]: This translation combines the construction presented here with the 'symmetric' translation LTL→LDBA based on the 'Master-Theorem' presented in [9,34], which treats least- (**F**, **U**, **M**) and greatest-fixed-point operators (**G**, **W**, **R**) symmetrically.

LD_p `ltl2dpa (portfolio)`[7]: Here, we combine the previous translations with a portfolio of translations for fragments that directly yield DPAs [9,34,35]. This portfolio approach is important in comparison to the configuration N_1 where similar steps are taken. Moreover, since complementation of DPAs is trivial, this configuration translates both the formula and its negation to DPAs A_φ, $A_{\neg\varphi}$, constructs the complement $\overline{A_{\neg\varphi}}$, and picks the smaller of A_φ and $\overline{A_{\neg\varphi}}$.

Table 1 Parametrised formulas set

$\chi_{1,n}$	$= (\dots((a_1 \mathbf{U} a_2)\mathbf{U} a_3)\dots \mathbf{U} a_{n+1})$
$\chi_{2,n}$	$= a_1 \mathbf{U}(a_2 \mathbf{U}(\dots(a_n \mathbf{U} a_{n+1})\dots))$
$\chi_{3,n}$	$= \mathbf{G}(a_1 \rightarrow a_1 \mathbf{U}(\dots(a_n \wedge a_n \mathbf{U} a_{n+1})))$
$\chi_{4,n}$	$= \bigwedge_{i=1}^{n}(\mathbf{F} a_i \vee \mathbf{G} a_{i+1})$
$\chi_{5,n}$	$= \bigwedge_{i=1}^{n} \mathbf{FG} a_i$
$\chi_{6,n}$	$= \bigwedge_{i=1}^{n} \mathbf{GF} a_i$
$\chi_{7,n}$	$= \left(\bigwedge_{i=1}^{n} \mathbf{GF} a_i\right) \rightarrow \mathbf{GF} b$
$\chi_{8,n}$	$= \left(\bigwedge_{i=1}^{n} \mathbf{GF} a_i\right) \leftrightarrow \mathbf{GF} b$
$\chi_{9,n}$	$= \bigwedge_{i=1}^{n}(\mathbf{GF} a_i \vee \mathbf{FG} a_{i+1})$
$\chi_{10,n}$	$= \mathbf{GF}(a \leftrightarrow \mathbf{X}^n a)$
$\chi_{11,n}$	$= \bigvee_{i=0}^{n} \mathbf{FG}((\neg)^i a \vee \mathbf{X}^i b)$

Input Formula Sets. We base the evaluation on two sets of formulas: the first set consists of the well-known 'Dwyer'-*patterns* [6] that collects 55 LTL formulas specifying common properties; the second set is obtained by instantiating the 11 *parametrised* formulas from Table 1. These families are partly taken from [14,29,38] or are simple combinations of **U**, **GF**, and **FG** formulas[8]. The second set of formulas is useful to isolate and analyse strong- and weak points of the compared translations. Furthermore, we abstained from using randomly generated formulas, because in our experience it is unclear what this implies for practice, since formulas from real-world examples usually have a high degree of structure compared to randomly generated formulas.

The formula sets are obtained by executing `genltl`[9] with the corresponding parameters. Each formula and its negation is then added to the set of formulas. We take the following steps to reduce the influence of specific simplification rules and to remove (close to) duplicate entries: first, we bring formulas into negation normal form; second, we apply a standard set of LTL simplification rules [1,11,29,33,37] with the goal to neutralise the effect LTL simplifier in the evaluation; third, we normalise the atomic propositions and

[4] We use the CLI flags: `-parity -deterministic`.

[5] We use the recommended CLI flags: `-k -j -t -i -r -o -m -d -u2`.

[6] We evaluate the state of the implementation defined by commit `cb12b3d6479555c2201e9cd190496d3d1fbe524a`.

[7] The source-code of all these tools is located in the repository of [20] and we evaluate the state of the implementation defined by commit `69c6557141b9d29ab0f37405c45fb75fa7f8608d`.

[8] This collection of parametrised formulas has been used before in [34].

[9] `genltl` is a component of `Spot` [5] to generate LTL formulas from existing patterns. We use the version `genltl (spot) 2.7.2`.

remove formulas that are equal modulo renaming of atomic propositions.

As a consequence, the number of formulas we consider is less than the number of formulas of the corresponding original publication. For example, [6] lists 55 formulas, but we remove six entries: e.g., only one of $\mathbf{G}a$, $\mathbf{G}\neg a$, and $\mathbf{F}a$ is added to the formula set. Note that we always evaluate the translation also on the negation of each formula. However, we do not remove duplicates across the two formula sets.

5.2 Results

The measured automata sizes for the LTL formulas are listed in Tables 2 and 4. We refer by φ to formulas of the pattern set, and by χ to formulas of the parametrised set. Further, we write $\overline{\varphi}$ instead of $\neg\varphi$. We sort the rows of the table by the difference in the orders of magnitude of sizes yielded by the considered configurations. More precisely, we compute $\frac{max}{min}$ for each row, where min refers to the number of states of the smallest automaton and max refers to the number of states of the largest automaton, and sort in descending order. In the main body of the paper, we list only the top 10 rows according to this order to highlight the most interesting differences. The remaining results are located in Appendix B.

5.3 Discussion

Table 2 and Table 4, which contain the formulas with the largest differences in size, suggest the following conclusions.

Variants of the LDBA approach. There are several cases where LD_1 produces dramatically smaller automata than LD_2, e.g., the first four rows of Table 2. Nevertheless, there are also many cases where LD_2 is slightly smaller LD_1, e.g., the following rows in that table and most of the formulas of Table 4. Thus both techniques have their merit, with their ratio close to 1 and the asymmetric being 'safer' if only one is to be used.

Observe that the same behaviour occurs with the pair D_1 and D_2, reflecting the fact that this difference stems from the difference between the asymmetric and symmetric approaches. This pattern is already noticeable for the intermediate constructions, i.e, the sizes of the constructed LDBAs and DRAs, respectively. The geometric averages for this intermediate step are 6.16, 5.68, 4.75, and 4.80 on the patterns set for LD_1, LD_2, D_1, and D_2, respectively, and 7.47, 7.94, 4.85, and 5.51 on the parametrised set. The complete results are located in "Appendix B".

The portfolio approach LD_p yields not only the smaller of the two results, but its dedicated constructions tailored to special fragments yield yet smaller LDBAs and correspondingly

Table 2 This table displays the results for the 'Dwyer'-patterns set. The table list number of states, followed by the number of colours (if larger than 1) and is sorted descending in regards to the largest difference in order of magnitude differences, as explained in the text. The results for the remaining 88 formulas are located in Appendix B. We write $\frac{1}{n}\Sigma$, σ, $\sqrt[n]{\Pi}$, and med., for the average, the standard deviation, the geometric average, and the median, respectively, for the number of states considering the whole data set

LTL	N_1	N_2	D_1	D_2	LD_1	LD_2	LD_p
φ_{49}	12(3)	19(4)	20(6)	1588(12)	15(7)	838(12)	15(7)
φ_{44}	10(3)	17(4)	22(6)	437(12)	15(7)	492(12)	15(7)
φ_{39}	17	8(2)	85(8)	256(6)	175(8)	385(8)	10(4)
φ_{14}	6	6(2)	12(4)	70(4)	7(4)	172(10)	7(4)
φ_{33}	4	4(2)	29(6)	6(4)	13(4)	6(2)	6(2)
$\overline{\varphi_{44}}$	9(3)	25(5)	30(6)	17(6)	58(9)	17(6)	16(7)
φ_{34}	3	3(2)	17(6)	13(4)	19(6)	13(4)	6(2)
$\overline{\varphi_{49}}$	15(3)	36(5)	39(6)	23(6)	74(9)	23(6)	16(7)
φ_{28}	4	4(2)	8(4)	20(6)	6(2)	18(4)	4(2)
$\overline{\varphi_{39}}$	6	22(4)	22(6)	17(5)	14(3)	10(4)	10(4)
$\frac{1}{n}\Sigma$	4.49	5.34	7.21	29.01	8.26	24.36	4.61
σ	2.94	5.16	10.31	166.13	19.35	104.41	2.84
$\sqrt[n]{\Pi}$	3.91	4.26	4.87	5.14	4.86	5.21	4.03
med.	4.0	4.0	4.0	4.0	4.0	4.0	4.0
φ_{14}	$\mathbf{G}(a \vee (\overline{b} \wedge \overline{c})\mathbf{U}(c \vee (b \wedge \overline{c})\mathbf{U}(c \vee (\overline{b} \wedge \overline{c})\mathbf{U}(c \vee (b \wedge \overline{c})\mathbf{U}(c \vee \mathbf{G}b \vee \overline{b}\mathbf{W}c)))))$						
φ_{28}	$\mathbf{G}(a \vee \mathbf{G}\overline{b} \vee c\mathbf{U}(b \vee (c \wedge d \wedge \mathbf{X}(c\mathbf{U}e))))$						
φ_{33}	$\mathbf{G}(a \vee \mathbf{G}\overline{b} \vee (b \vee c \vee \mathbf{X}(b\mathbf{R}(b \vee d)))\mathbf{U}(b \vee e))$						
φ_{34}	$\mathbf{G}(a \vee \mathbf{G}(b \vee \mathbf{X}\mathbf{G}c) \vee (b \vee d \vee \mathbf{X}(d\mathbf{R}(c \vee d)))\mathbf{U}(d \vee e))$						
φ_{39}	$\mathbf{G}(a \vee (b \vee \mathbf{X}(c\mathbf{R}\overline{d}) \vee \mathbf{X}(\overline{c}\mathbf{U}(d \wedge \mathbf{F}e)))\mathbf{U}(c \vee \mathbf{G}(b \vee \mathbf{X}(c\mathbf{R}\overline{d}) \vee \mathbf{X}(\overline{c}\mathbf{U}(d \wedge \mathbf{F}e)))))$						
φ_{44}	$\mathbf{G}(a \vee (b \vee \overline{c}\mathbf{U}(\overline{c} \wedge d \wedge \mathbf{X}(\overline{c}\mathbf{U}e)))\mathbf{U}(c \vee \mathbf{G}(b \vee (d \wedge \mathbf{X}\mathbf{F}e))))$						
φ_{49}	$\mathbf{G}(a \vee (b \vee \overline{c}\mathbf{U}(\overline{c} \wedge d \wedge e \wedge \mathbf{X}((\overline{c} \wedge e)\mathbf{U}f)))\mathbf{U}(c \vee \mathbf{G}(b \vee (d \wedge e \wedge \mathbf{X}(e\mathbf{U}f)))))$						

Table 3 Excerpt of Table 2 highlighting the effect of negation and complementation used in the portfolio approach

LTL	N_1	N_2	D_1	D_2	LD_1	LD_2	LD_p
φ_{39}	17	8(2)	85(8)	256(6)	175(8)	385(8)	10(4)
$\overline{\varphi_{39}}$	6	22(4)	22(6)	17(5)	14(3)	10(4)	10(4)
φ_{44}	10(3)	17(4)	22(6)	437(12)	15(7)	492(12)	15(7)
$\overline{\varphi_{44}}$	9(3)	25(5)	30(6)	17(6)	58(9)	17(6)	16(7)
φ_{49}	12(3)	19(4)	20(6)	1588(12)	15(7)	838(12)	15(7)
$\overline{\varphi_{49}}$	15(3)	36(5)	39(6)	23(6)	74(9)	23(6)	16(7)

Table 4 This table displays the results for the parametrised set (Table 1). The table is structured as Table 2 and the results for the remaining 56 formulas are located in Appendix B

LTL	N_1	N_2	D_1	D_2	LD_1	LD_2	LD_p
$\chi_{9,4}$	2937(5)	3220(6)	1392(10)	196(10)	471(10)	337(14)	24(6)
$\overline{\chi_{6,5}}$	5	472(3)	120(2)	120(2)	5(2)	5(2)	5(2)
$\overline{\chi_{8,4}}$	189(2)	1562(4)	155(11)	201(5)	262(3)	201(3)	20(4)
$\chi_{7,4}$	91(2)	241(3)	24(4)	24(4)	4(4)	4(4)	4(3)
$\chi_{9,3}$	154(4)	152(6)	36(8)	26(8)	51(8)	37(10)	6(5)
$\overline{\chi_{11,4}}$	15	314(2)	62(3)	15(3)	15(3)	15(3)	15(2)
$\overline{\chi_{6,4}}$	4	89(3)	24(2)	24(2)	4(2)	4(2)	4(2)
$\chi_{7,3}$	24(2)	66(3)	6(4)	6(4)	3(4)	3(4)	3(3)
$\chi_{11,4}$	15	49(3)	244(2)	244(2)	15(2)	15(2)	15(2)
$\chi_{8,4}$	77(2)	299(3)	66(4)	56(4)	20(4)	56(4)	20(4)
$\frac{1}{n}\Sigma$	61.47	115.45	42.53	24.03	23.47	20.11	8.91
σ	358.12	435.39	171.80	47.10	65.63	48.95	9.13
$\sqrt[n]{\Pi}$	8.16	17.46	9.32	8.53	7.36	7.14	5.71
med.	7.0	16.5	6.5	6.5	5.5	5.0	5.0

also DPAs for several formulas. Overall, not surprisingly, a portfolio approach entails a considerable advantage.

Safra versus IAR versus LDBA determinisation Our LDBA determinisation in a portfolio configuration (LD_p) is on-par with N_1 on the pattern set, the average ratio being 103%, and takes the lead in the parametrised setting, the average ratio being 70% there.

For LD_1 and LD_2, without post-processing and portfolio techniques, the difference to N_1 grows (with the ratios 124% and 133%, respectively). Yet, on the parametrised set they are still better than the the portfolio N_1 (with 90% and 87%, respectively).

One of the reasons for the discrepancy between comparisons on the pattern set and on the parametrised set is that the parametrised set contains several 'simpler' formulas that are recognisable by a deterministic Büchi or deterministic co-Büchi automaton, as indicated by N_1 only needing a single colour. In these 'simple' cases, several techniques are known, and implemented in N_1, to reduce the number of states in the automata.

The other Safra-based approach N_2 tends to yield larger automata than N_1.

Comparing the IAR and LDBA-determinisation approaches, interestingly, there are significant differences in both directions on many formulas; yet the ratios are close to 1 on the pattern set, showing they are quite incomparable. However, the latter takes the lead on the parametrised set.

Summary The portfolio approaches are the clear winners. While N_1 may produce slightly smaller automata in many cases, LD_p produces in several cases significantly smaller automata. Both approaches are practically used in the LTL synthesis: N_1 is used in `ltlsynt` [15] and a variant of LD_p is used in `Strix` [27]. We leave open the question, whether implementing a portfolio for the IAR approach would also yield a competitive configuration.

Finally, having pointed out the differences, it is important to keep in mind that, for a substantial part of the formulas, all participating tools yield small automata, not differing dramatically, as one can see by inspecting "Appendix B".

6 Conclusion

We have presented a simple, 'Safraless', and asymptotically optimal translation from LTL and LDBA to DPA.

Furthermore, the translation is suitable for an on-the-fly implementation and deployment in the LTL synthesis, which has been successfully demonstrated by `Strix` [27], the winner of the LTL-synthesis track of SyntComp 2018 [16] and 2019.[10]

A Proof of Proposition 4

We start by recalling the LTL→LDBA translation of [36].

Preliminaries. The translation assumes that formulas are in *negation normal form*, given by the syntax

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi \mathbf{U} \varphi$$

where a belongs to a finite set of atomic propositions. Every formula over the usual syntax of LTL (with negation and the **X** and **U** operators) can be normalised with linear blowup if formulas are represented by their syntax DAGs, where two occurrences of the same subformula are represented by the same node.

We recall the *af* function introduced in [7,36], and some of its properties. Let σ be a letter. The formula $af(\varphi, \sigma)$, read "φ after σ", is inductively defined as follows [7,36]:

$$\begin{aligned}
af(\mathbf{tt}, \sigma) &= \mathbf{tt} \\
af(\mathbf{ff}, \sigma) &= \mathbf{ff} \\
af(a, \sigma) &= \begin{cases} \mathbf{tt} & \text{if } a \in \sigma \\ \mathbf{ff} & \text{if } a \notin \sigma \end{cases} \\
af(\neg a, \sigma) &= \begin{cases} \mathbf{ff} & \text{if } a \in \sigma \\ \mathbf{tt} & \text{if } a \notin \sigma \end{cases} \\
af(\varphi \wedge \psi, \sigma) &= af(\varphi, \sigma) \wedge af(\psi, \sigma) \\
af(\varphi \vee \psi, \sigma) &= af(\varphi, \sigma) \vee af(\psi, \sigma) \\
af(\mathbf{X}\varphi, \sigma) &= \varphi \\
af(\mathbf{G}\varphi, \sigma) &= af(\varphi, \sigma) \wedge \mathbf{G}\varphi \\
af(\mathbf{F}\varphi, \sigma) &= af(\varphi, \sigma) \vee \mathbf{F}\varphi \\
af(\varphi \mathbf{U} \psi, \sigma) &= af(\psi, \sigma) \vee (af(\varphi, \sigma) \wedge \varphi \mathbf{U} \psi)
\end{aligned}$$

Furthermore, we define: $af(\varphi, \epsilon) = \varphi$, and $af(\varphi, \sigma w) = af(af(\varphi, \sigma), w)$ for every letter σ and every finite word w. The function *af* has the following two properties [7,36] for every formula φ, finite word v, and ω-word w:

(i) $\sigma w \models \varphi$ iff $w \models af(\varphi, \sigma)$.
(ii) $af(\varphi, \sigma)$ is a boolean combination of subformulas of φ.

A formula is *proper* if it is neither a conjunction nor a disjunction. The propositional formula φ_P of a formula φ is the result of substituting every maximal proper subformula ψ of φ by a propositional variable x_ψ. For example, if $\varphi = \mathbf{X}b \vee (\mathbf{G}(a \vee \mathbf{X}b) \wedge \mathbf{X}b)$ with $\psi_1 = \mathbf{X}b$ and $\psi_2 = \mathbf{G}(a \vee \mathbf{X}b)$, then $\varphi_P = x_{\psi_1} \vee (x_{\psi_2} \wedge x_{\psi_1})$. Two formulas φ, φ' are *propositionally equivalent*, denoted $\varphi \equiv_P \varphi'$, if φ_P and φ'_P are equivalent. So, for example, $\mathbf{X}b$ is propositionally equivalent to $\mathbf{X}b \vee (\mathbf{G}(a \vee \mathbf{X}b) \wedge \mathbf{X}b)$. Observe that propositional equivalence implies equivalence, but the contrary does not hold. For example, $\mathbf{F}a \wedge \mathbf{G}a$ and $\mathbf{G}a$ are equivalent, but not propositionally equivalent.

The states of the LDBA for an LTL formula are equivalence classes of formulas (or tuples thereof) with respect to propositional equivalence. However, we abuse language and write that the states are formulas or tuples of formulas.

Translating LTL to LDBA. Fix a formula φ. We describe the LDBA $\mathcal{A}_\varphi$. We use $\varphi = c \vee \mathbf{XG}(a \vee \mathbf{F}b)$ as running example. We abbreviate $\psi := (a \vee \mathbf{F}b)$, and write $\varphi = c \vee \mathbf{XG}\psi$. The LDBA $\mathcal{A}_\varphi$ is shown in Fig. 8.

The LDBA $\mathcal{A}_\varphi$ consists of two deterministic components, called the *initial* and *accepting* components, and denoted $\mathcal{A}_{in}$ and $\mathcal{A}_{ac}$, respectively—in Fig. 8 they are shown above and below the dashed line. The accepting component $\mathcal{A}_{ac}$ is the union (defined componentwise for states, transitions, and accepting states) of subcomponents $\mathcal{A}_\mathcal{G}$, one for each set $\mathcal{G}$ of **G**-subformulas of φ–that is, if φ has n different **G**-subformulas, then $\mathcal{A}_{ac}$ is the union of 2^n subcomponents). Transitions of $\mathcal{A}_\varphi$ labeled by an alphabet letter connect either two states of $\mathcal{A}_{in}$, or two states of the same subcomponent of $\mathcal{A}_{ac}$. Further, for each state q of $\mathcal{A}_{in}$ and each set $\mathcal{G}$ there is an ϵ-transition leading from q to a state of $\mathcal{A}_\mathcal{G}$.

Initial component $\mathcal{A}_{in}$: Define the set of formulas *reachable* from φ as $Reach(\varphi) = \{\psi \mid \exists w.\ \psi = af(\varphi, w)\}$. The set of states of $\mathcal{A}_{in}$ is $Reach(\varphi)$. The initial state is φ. The transition function δ_{in} is given by $\delta_{in}(\psi, a) = af(\psi, a)$. Intuitively, $\mathcal{A}_{in}$ monitors the formula that has to hold at the current moment for φ to hold at the beginning.

Accepting component $\mathcal{A}_{ac}$: The accepting component $\mathcal{A}_{ac}$ is the union of subcomponents $\mathcal{A}_\mathcal{G}$, one for each $\mathcal{G} \subseteq \mathbb{G}(\varphi)$.

Let $\mathbb{G}(\varphi)$ denote the set of all **G**-subformulas of φ. Given a set $\mathcal{G} \subseteq \mathbb{G}(\varphi)$ and a formula ψ, we write $\psi[\mathcal{G}]$ as an abbreviation for $\psi[\mathcal{G}, \mathbb{G}(\varphi) \setminus \mathcal{G}]$, i.e., for the result of substituting **tt** for each maximal occurrence of a formula of $\mathcal{G}$ in ψ, and **ff** for each maximal occurrence of a formula of $\mathbb{G}(\varphi) \setminus \mathcal{G}$ in ψ. For example, if $\mathcal{G} = \{\mathbf{G}(a \vee \mathbf{G}b)\}$ then $\mathbf{G}b \vee \mathbf{X}(a \wedge \mathbf{G}(a \vee \mathbf{G}b))[\mathcal{G}] = \mathbf{ff} \vee \mathbf{X}(a \wedge \mathbf{tt}) \equiv \mathbf{X}a$.

Each subcomponent $\mathcal{A}_\mathcal{G}$ is a product of DBAs: One for the formula $\varphi[\mathcal{G}]$, and one for each formula of the form $\mathbf{G}(\psi[\mathcal{G}])$, where $\mathbf{G}\psi \in \mathcal{G}$. Observe that $\varphi[\mathcal{G}]$ is a **G**-free formula, and $\mathbf{G}(\psi[\mathcal{G}])$ does not have nested **G**s. For example, if $\mathcal{G} = \{\mathbf{G}(a \vee \mathbf{G}b)\}$, then $\mathcal{A}_\mathcal{G}$ is the product of three DBAs, one for $\varphi[\mathcal{G}]$, one for $\mathbf{G}(b[\mathcal{G}]) = \mathbf{G}b$, and a third one for $\mathbf{G}((a \vee \mathbf{G}b)[\mathcal{G}]) = \mathbf{G}(a \vee \mathbf{ff}) \equiv_P \mathbf{G}a$. We call the DBAs for $\varphi[\mathcal{G}]$ and $\mathbf{G}(\psi[\mathcal{G}])$ the *monitors*.

[10] http://www.syntcomp.org/syntcomp-2019-results/

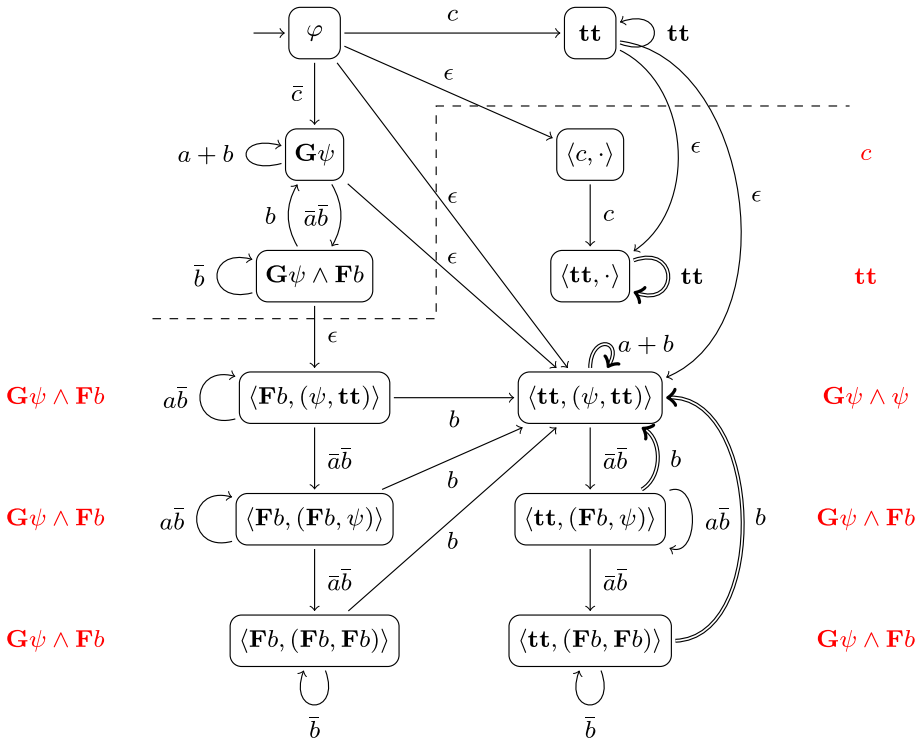

Fig. 8 Automaton $\mathcal{A}$ for $\varphi = c \vee \mathbf{XG}(a \vee \mathbf{F}b)$. The initial component is above the dashed line, the accepting component below

Monitor for $\varphi[\mathcal{G}]$. The set of states is $Reach(\varphi[\mathcal{G}])$, the transition function $\delta_{\varphi[\mathcal{G}]}$ is given by $\delta_{\varphi[\mathcal{G}]}(\psi, \sigma) = af(\psi, \sigma)$. The only final state is $\mathbf{tt}$. The initial state is left unspecified.

⋆ Lemma 2 of [36] shows that the $\varphi[\mathcal{G}]$-monitor accepts a word w from a state q iff w satisfies the formula q.

Monitor for $\mathbf{G}(\psi[\mathcal{G}])$. Let us abbreviate $\psi[\mathcal{G}]$ as ψ'. The monitor for $\mathbf{G}\psi'$ is the DBA $\mathcal{U}(\mathbf{G}\psi') = (2^{Ap}, Reach(\psi') \times Reach(\psi'), \delta, (\psi', \mathbf{tt}), F)$ where

– $\delta((\xi_1, \xi_2), \sigma) = \begin{cases} (af(\xi_2, \sigma) \wedge \psi', \mathbf{tt}) & \text{if } af(\xi_1, \sigma) \equiv_P \mathbf{tt} \\ (af(\xi_1, \sigma), af(\xi_2, \sigma) \wedge \psi') & \text{otherwise} \end{cases}$

– $F = \{((\xi_1, \xi_2), \sigma, p) \in Q \times 2^{Ap} \times Q \mid af(\xi_1, \sigma) \equiv_P \mathbf{tt}\}$

⋆ Lemma 5 of [36] proves that $\mathcal{U}(\mathbf{G}\psi')$ accepts a word w (from its initial state $(\psi', \mathbf{tt})$) iff $w \models \mathbf{G}\psi'$.

Product. Fix a set $\mathcal{G} = \{\mathbf{G}\psi_1, \ldots, \mathbf{G}\psi_n\} \subseteq \mathbb{G}$ of $\mathbf{G}$-subformulas of φ. For every index $1 \leq i \leq n$, let $\mathcal{U}_i = (2^{Ap}, Q_i, \delta_i, q_{0i}, F_i)$ be the monitor for $\mathbf{G}(\psi_i[\mathcal{G}])$. The *product* of these monitors is the generalised deterministic Büchi automaton

$$\mathcal{P}(\mathcal{G}) = (\, 2^{Ap},\ \prod_{i=1}^{n} Q_i,\ \prod_{i=1}^{n} \delta_i,\ (q_{01}, \ldots, q_{0n}),\ \{F_1', \ldots, F_n'\}\,)$$

where $(\,(q_1, \ldots, q_n), \sigma, (q, q_1', \ldots, q_n')\,)$ is a transition of F_i' iff $(q_i, \sigma, q_i') \in F_i$.

⋆ Lemma 5 of [36] proves that $\mathcal{P}(\mathcal{G})$ accepts w iff $w \models \mathbf{G}(\psi[\mathcal{G}])$ for all $\mathbf{G}\psi \in \mathcal{G}$.

Subcomponent $\mathcal{A}_\mathcal{G}$. The subcomponent is the product of the monitor for $\varphi[\mathcal{G}]$ and $\mathcal{P}(\mathcal{G})$:

$$\mathcal{A}_\mathcal{G} = (\, 2^{Ap}\ ,\ Reach(\varphi[\mathcal{G}]) \times \prod_{i=1}^{n} Q_i,$$

$$\delta_{\varphi[\mathcal{G}]} \times \prod_{i=1}^{n} \delta_i\ ,\ \{\{\} \times F_1', \ldots, \{\} \times F_n'\}\,)$$

⋆ We have: $\mathcal{A}_\mathcal{G}$ accepts a word w from the state $(\varphi'[\mathcal{G}], q_{01}, \ldots, q_{0n})$ iff $w \models \varphi'[\mathcal{G}] \wedge \mathbf{G}(\psi[\mathcal{G}])$.

Connecting ϵ**-transitions:** Finally, we describe the ϵ-transitions connecting the initial component $\mathcal{A}_{in}$ to the accepting component $\mathcal{A}_{ac}$. There is an ϵ-transition for each state φ' of $\mathcal{A}_{in}$ and each set $\mathcal{G} = \{\mathbf{G}\psi_1, \ldots, \mathbf{G}\psi_n\} \subseteq \mathbb{G}(\varphi)$. The transition is $\big(\varphi', \epsilon, (\,\varphi'[\mathcal{G}], \mathbf{G}(\psi_1[\mathcal{G}]), \ldots, \mathbf{G}(\psi_n[\mathcal{G}])\,)\big)$.

⋆ **Theorem 1** of [36] *proves that* $w \models \varphi$ iff $\mathcal{A}_\varphi$ *accepts* w.

This concludes the description of $\mathcal{A}_\varphi$.

Proof of Proposition 4 We start by generalizing Lemma 5 of [36] as follows:

Lemma 5 *$\mathcal{U}(\mathbf{G}\psi')$ accepts a word w from a state (ξ_1, ξ_2) iff $w \models \mathbf{G}\psi' \wedge \xi_1 \wedge \xi_2$.*

Proof We first claim that there is a formula ξ such that $\xi_1 \wedge \xi_2 \equiv \xi \wedge \psi'$. If $(\xi_1, \xi_2) = (\psi', \mathbf{tt})$, then $\xi = \mathbf{tt}$. Otherwise there is (ξ_1', ξ_2') and σ such that $\delta((\xi_1', \xi_2'), \sigma) = (\xi_1, \xi_2)$. By the definition of the transition function, either $\xi_1 = \eta_1 \wedge \psi'$ for some η_1, or $\xi_2 = \eta_2 \wedge \psi'$ for some η_2, and we can choose ξ accordingly.

By this result, we have $af(\xi_1 \wedge \xi_2, \sigma) \equiv af(\xi \wedge \psi', \sigma) \equiv af(\xi, \sigma) \wedge af(\psi', \sigma)$ for every state (ξ_1, ξ_2) and letter σ, and so in particular

$$af(\xi_1, \sigma) \wedge af(\xi_2, \sigma) \models af(\psi', \sigma) \tag{1}$$

We now prove the lemma. Let v be a finite word leading from the initial state $(\psi', \mathbf{tt})$ to (ξ_1, ξ_2). We proceed by induction of the length of v.

Basis. $v = \epsilon$. Then $(\xi_1, \xi_2) = (\psi', \mathbf{tt})$, and so $w \models \mathbf{G}\psi' \wedge \xi_1 \wedge \xi_2$ iff $w \models \mathbf{G}\psi' \wedge \psi' \equiv \mathbf{G}\psi'$. By Lemma 5 of [36] $w \models \mathbf{G}\psi'$ iff $\mathcal{U}(\mathbf{G}\psi')$ accepts w from $(\psi', \mathbf{tt})$, and we are done.

Step. $v = v'\sigma$ for some word v' and letter σ. Then there is a state (ξ_i', ξ_2') such that $\delta((\psi', \mathbf{tt}), v') = (\xi_1', \xi_2')$ and $\delta((\xi_1', \xi_2'), v') = (\xi_1, \xi_2)$. By induction hypothesis a word w is accepted from (ξ_1', ξ_2') iff $w \models \mathbf{G}\psi' \wedge \xi_1' \wedge \xi_2'$. We consider two cases.

If $af(\xi_1', \sigma) \equiv_P \mathbf{tt}$, then by the definition of δ we have $\delta((\xi_1', \xi_2'), \sigma) = (af(\xi_2', \sigma) \wedge \psi', \mathbf{tt})$. It follows:

$\mathcal{U}(\mathbf{G}\psi')$ accepts w from(ξ_1, ξ_2)
iff (determinism)
$\mathcal{U}(\mathbf{G}\psi')$ accepts σw from(ξ_1', ξ_2')
iff (induction hypothesis)
$\sigma w \models \mathbf{G}\psi' \wedge \xi_1' \wedge \xi_2'$
iff (fundamental property of af)
$w \models af(\mathbf{G}\psi' \wedge \xi_1' \wedge \xi_2', \sigma)$
iff (definition of af)
$w \models \mathbf{G}\psi' \wedge af(\psi'\sigma) \wedge af(\xi_1', \sigma) \wedge af(\xi_2', \sigma)$
iff (Equation 1)
$w \models \mathbf{G}\psi' \wedge af(\xi_1', \sigma) \wedge af(\xi_2', \sigma)$

We conclude the proof by showing $\mathbf{G}\psi' \wedge af(\xi_1', \sigma) \wedge af(\xi_2', \sigma) \equiv \mathbf{G}\psi' \wedge \xi_1 \wedge \xi_2$. It suffices to prove $\xi_1 \wedge \xi_2 \equiv af(\xi_1', \sigma) \wedge af(\xi_2', \sigma) \wedge \psi'$. Consider two cases:

- $af(\xi_1', \sigma) \equiv_P \mathbf{tt}$. Then, by the definition of $\mathcal{U}(\mathbf{G}\psi')$, we have $\xi_1 = af(\xi_2', \sigma) \wedge \psi'$ and $\xi_2 = \mathbf{tt}$, and we are done.
- $af(\xi_1', \sigma) \not\equiv_P \mathbf{tt}$. Then, by the definition of $\mathcal{U}(\mathbf{G}\psi')$, we have $\xi_1 = af(\xi_1', \sigma)$ and $\xi_2 = af(\xi_1', \sigma) \wedge \psi'$, and we are done. □

We can now proceed to prove Proposition 4.

Proposition 4 *For every LTL formula φ, every state s of the LDBA of [36] for φ can be 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.*

Further, $label(s)$ can be computed in linear time from the descriptor of s.

Proof Recall the two properties of the af function. For every formula φ, finite word v, and ω-word w:

(i) $vw \models \varphi$ iff $w \models af(\varphi, v)$.
(ii) $af(\varphi, v)$ is a boolean combination of subformulas of φ. Therefore, every formula of $Reach(\varphi)$ is a boolean combination of subformulas of φ.

Let s be a state of $\mathcal{A}_{in}$, and let v be any finite word leading from s_0 to s. By Theorem 1 of [36], we have $L(s_0) = L(\varphi)$. By (i), $\mathsf{L}(af(\varphi, v)) = \{w \mid vw \in \mathsf{L}(\varphi)\}$. Since $\mathcal{A}_{in}$ is deterministic, $\mathsf{L}(s) = \{w \mid vw \in \mathsf{L}(s_0)\} = \{w \mid vw \in \mathsf{L}(\varphi)\} = \mathsf{L}(af(\varphi, v))$. So we can take $label(s) = s$.

We consider now the case that s belongs to $\mathcal{A}_{ac}$. Then there is a set $\mathcal{G} = (\mathbf{G}\psi_1, \ldots, \mathbf{G}\psi_n)$ such that s belongs to $\mathcal{A}_{\mathcal{G}}$. By the definition of $\mathcal{A}_{\mathcal{G}}$ as product of DBAs, s is of the form $(\varphi'[\mathcal{G}], (\xi_{11}, \xi_{21}), \ldots, (\xi_{1n}, \xi_{2n}))$, where $\varphi'[\mathcal{G}]$ is a state of the monitor for $\varphi[\mathcal{G}]$, and (ξ_{1i}, ξ_{2i}) is a state of the monitor for $\mathbf{G}(\psi_i[\mathcal{G}])$. Further, the words recognised from s are those simultaneously recognized from $\varphi'[\mathcal{G}]$, $(\xi_{11}, \xi_{21}), \ldots, (\xi_{1n}, \xi_{2n})$ in their respective automata. By Lemma 5, the words recognised from s are those satisfying $\varphi'[\mathcal{G}] \wedge \xi_{11} \wedge \xi_{21} \wedge \ldots \wedge \xi_{1n} \wedge \xi_{2n}$. We choose $label(s)$ as this formula. It remains to show that each conjunct of $label(s)$ is a boolean combination of formulas of $\mathsf{sf}(\varphi)[\mathcal{G}]$.

- By the definition of the monitor for $\varphi[\mathcal{G}]$, the formula $\varphi'[\mathcal{G}]$ belongs to $Reach(\varphi[\mathcal{G}])$, and by (ii) we are done.
- By the definition of the monitor for $\mathbf{G}(\psi_i[\mathcal{G}])$, the formulas ξ_{1i} and ξ_{2i} belong to $Reach(\psi_i[\mathcal{G}])$. By (ii), they are boolean combinations of subformulas of $\psi_i[\mathcal{G}]$. Since ψ_i is a subformula of φ, they are also boolean combinations of subformulas of $\varphi[\mathcal{G}]$, and so a boolean combination of formulas of $\mathsf{sf}(\varphi)[\mathcal{G}]$. □

B Additional experimental results

We list in Table 5 the complete and normalised 'Dwyer'-pattern formula set. Tables 6 and 7 contain missing entries from Tables 2, and 8 contains entries missing from Table 4, respectively. Moreover, we list the sizes of the intermediate automata in Tables 9 and 10.

Table 5 Complete and normalised 'Dwyer'-pattern formula set

φ_1	$\mathbf{G}a$
φ_2	$\mathbf{G}\overline{a} \vee b\mathbf{U}a$
φ_3	$\mathbf{G}(a \vee \mathbf{G}b)$
φ_4	$\mathbf{G}(a \vee b \vee \mathbf{G}\overline{b} \vee c\mathbf{U}b)$
φ_5	$\mathbf{G}(a \vee b \vee c\mathbf{W}b)$
φ_6	$a\mathbf{W}(a \wedge b)$
φ_7	$\mathbf{G}\overline{a} \vee \mathbf{F}(a \wedge \mathbf{F}b)$
φ_8	$\mathbf{G}(a \vee b \vee \overline{b}\mathbf{W}(\overline{b} \wedge c))$
φ_9	$\mathbf{G}(a \vee b \vee \overline{b}\mathbf{U}(\overline{b} \wedge c))$
φ_{10}	$\overline{a}\mathbf{W}(a\mathbf{W}(\overline{a}\mathbf{W}(a\mathbf{W}(\mathbf{G}\overline{a}))))$
φ_{11}	$\mathbf{G}\overline{a} \vee (\overline{a} \wedge \overline{b})\mathbf{U}(a \vee (\overline{a} \wedge b)\mathbf{U}(a \vee (\overline{a} \wedge \overline{b})\mathbf{U}(a \vee (\overline{a} \wedge b)\mathbf{U}(a \vee \overline{b}\mathbf{U}a))))$
φ_{12}	$\overline{a}\mathbf{W}(a \wedge \overline{b}\mathbf{W}(b\mathbf{W}(\overline{b}\mathbf{W}(b\mathbf{W}(\mathbf{G}\overline{b})))))$
φ_{13}	$\mathbf{G}(a \vee \mathbf{G}\overline{b} \vee (\overline{b} \wedge \overline{c})\mathbf{U}(b \vee (\overline{b} \wedge c)\mathbf{U}(b \vee (\overline{b} \wedge \overline{c})\mathbf{U}(b \vee (\overline{b} \wedge c)\mathbf{U}(b \vee \overline{c}\mathbf{U}b)))))$
φ_{14}	$\mathbf{G}(a \vee (\overline{b} \wedge \overline{c})\mathbf{U}(c \vee (b \wedge \overline{c})\mathbf{U}(c \vee (\overline{b} \wedge \overline{c})\mathbf{U}(c \vee (b \wedge \overline{c})\mathbf{U}(c \vee \mathbf{G}b \vee \overline{b}\mathbf{W}c)))))$
φ_{15}	$a\mathbf{W}b$
φ_{16}	$\mathbf{G}\overline{a} \vee b\mathbf{U}(a \vee c)$
φ_{17}	$\mathbf{G}\overline{a} \vee \mathbf{F}(a \wedge b\mathbf{W}c)$
φ_{18}	$\mathbf{G}(a \vee b \vee \mathbf{G}\overline{b} \vee c\mathbf{U}(b \vee d))$
φ_{19}	$\mathbf{G}(a \vee b \vee c\mathbf{W}(b \vee d))$
φ_{20}	$\mathbf{G}(a \vee \mathbf{F}b)$
φ_{21}	$\mathbf{G}\overline{a} \vee (b \vee \overline{a}\mathbf{U}(\overline{a} \wedge c))\mathbf{U}a$
φ_{22}	$\mathbf{G}(a \vee \mathbf{G}(b \vee \mathbf{F}c))$
φ_{23}	$\mathbf{G}(a \vee b \vee \mathbf{G}\overline{b} \vee (c \vee \overline{b}\mathbf{U}(\overline{b} \wedge d))\mathbf{U}b)$
φ_{24}	$\mathbf{G}(a \vee b \vee (c \vee \overline{b}\mathbf{U}(\overline{b} \wedge d))\mathbf{W}b)$
φ_{25}	$a\mathbf{W}(a \wedge b \wedge \mathbf{X}(a\mathbf{U}c))$
φ_{26}	$\mathbf{G}\overline{a} \vee b\mathbf{U}(a \vee (b \wedge c \wedge \mathbf{X}(b\mathbf{U}d)))$
φ_{27}	$a\mathbf{W}(a \vee b\mathbf{W}(b \wedge c \wedge \mathbf{X}(b\mathbf{U}d)))$
φ_{28}	$\mathbf{G}(a \vee \mathbf{G}\overline{b} \vee c\mathbf{U}(b \vee (c \wedge d \wedge \mathbf{X}(c\mathbf{U}e))))$
φ_{29}	$\mathbf{G}(a \vee b\mathbf{W}(c \vee (b \wedge d \wedge \mathbf{X}(b\mathbf{U}e))))$
φ_{30}	$a\mathbf{U}b \vee \mathbf{G}(a \vee \mathbf{X}\mathbf{G}c)$
φ_{31}	$\mathbf{G}\overline{a} \vee (a \vee b \vee \mathbf{X}(a\mathbf{R}(a \vee c)))\mathbf{U}(a \vee d)$
φ_{32}	$\overline{a}\mathbf{W}(a \wedge (b\mathbf{U}c \vee \mathbf{G}(b \vee \mathbf{X}\mathbf{G}d)))$
φ_{33}	$\mathbf{G}(a \vee \mathbf{G}\overline{b} \vee (b \vee c \vee \mathbf{X}(b\mathbf{R}(b \vee d)))\mathbf{U}(b \vee e))$
φ_{34}	$\mathbf{G}(a \vee \mathbf{G}(b \vee \mathbf{X}\mathbf{G}c) \vee (b \vee d \vee \mathbf{X}(d\mathbf{R}(c \vee d)))\mathbf{U}(d \vee e))$
φ_{35}	$\mathbf{G}(a \vee \mathbf{X}\mathbf{G}\overline{b} \vee \mathbf{X}\mathbf{F}(b \wedge \mathbf{F}c))$
φ_{36}	$\mathbf{G}\overline{a} \vee (b \vee \mathbf{X}(a\mathbf{R}\overline{c}) \vee \mathbf{X}(\overline{a}\mathbf{U}(c \wedge \mathbf{F}d)))\mathbf{U}a$
φ_{37}	$\mathbf{G}(a \vee \mathbf{G}(b \vee \mathbf{X}\mathbf{G}\overline{c} \vee \mathbf{X}\mathbf{F}(c \wedge \mathbf{F}d)))$
φ_{38}	$\mathbf{G}(a \vee \mathbf{G}\overline{b} \vee (c \vee \mathbf{X}(b\mathbf{R}\overline{d}) \vee \mathbf{X}(\overline{b}\mathbf{U}(d \wedge \mathbf{F}e)))\mathbf{U}b)$
φ_{39}	$\mathbf{G}(a \vee (b \vee \mathbf{X}(c\mathbf{R}\overline{d}) \vee \mathbf{X}(\overline{c}\mathbf{U}(d \wedge \mathbf{F}e)))\mathbf{U}(c \vee \mathbf{G}(b \vee \mathbf{X}(c\mathbf{R}\overline{d}) \vee \mathbf{X}(\overline{c}\mathbf{U}(d \wedge \mathbf{F}e)))))$
φ_{40}	$\mathbf{G}(a \vee \mathbf{F}(b \wedge \mathbf{X}\mathbf{F}c))$
φ_{41}	$\mathbf{G}\overline{a} \vee (b \vee \overline{a}\mathbf{U}(\overline{a} \wedge c \wedge \mathbf{X}(\overline{a}\mathbf{U}d)))\mathbf{U}a$
φ_{42}	$\mathbf{G}(a \vee \mathbf{G}(b \vee (c \wedge \mathbf{X}\mathbf{F}d)))$
φ_{43}	$\mathbf{G}(a \vee \mathbf{G}\overline{b} \vee (c \vee \overline{b}\mathbf{U}(\overline{b} \wedge d \wedge \mathbf{X}(\overline{b}\mathbf{U}e)))\mathbf{U}b)$
φ_{44}	$\mathbf{G}(a \vee (b \vee \overline{c}\mathbf{U}(\overline{c} \wedge d \wedge \mathbf{X}(\overline{c}\mathbf{U}e)))\mathbf{U}(c \vee \mathbf{G}(b \vee (d \wedge \mathbf{X}\mathbf{F}e))))$
φ_{45}	$\mathbf{G}(a \vee \mathbf{F}(b \wedge c \wedge \mathbf{X}(c\mathbf{U}d)))$
φ_{46}	$\mathbf{G}\overline{a} \vee (b \vee \overline{a}\mathbf{U}(\overline{a} \wedge c \wedge d \wedge \mathbf{X}((\overline{a} \wedge d)\mathbf{U}e)))\mathbf{U}a$
φ_{47}	$\mathbf{G}(a \vee \mathbf{G}(b \vee (c \wedge d \wedge \mathbf{X}(d\mathbf{U}e))))$
φ_{48}	$\mathbf{G}(a \vee \mathbf{G}\overline{b} \vee (c \vee \overline{b}\mathbf{U}(\overline{b} \wedge d \wedge e \wedge \mathbf{X}((\overline{b} \wedge e)\mathbf{U}f)))\mathbf{U}b)$
φ_{49}	$\mathbf{G}(a \vee (b \vee \overline{c}\mathbf{U}(\overline{c} \wedge d \wedge e \wedge \mathbf{X}((\overline{c} \wedge e)\mathbf{U}f)))\mathbf{U}(c \vee \mathbf{G}(b \vee (d \wedge e \wedge \mathbf{X}(e\mathbf{U}f)))))$

Table 6 This table is a continuation of Table 2

LTL	N_1	N_2	D_1	D_2	LD_1	LD_2	LD_p
φ_{13}	7	7(2)	22(4)	7(4)	7(2)	25(10)	7(2)
$\overline{\varphi_{45}}$	4	6(3)	6(4)	5(2)	15(6)	5(2)	4(2)
φ_{43}	4	4(2)	6(4)	12(6)	4(2)	12(4)	4(2)
φ_{48}	4	4(2)	6(4)	12(6)	4(2)	12(4)	4(2)
$\overline{\varphi_{38}}$	8	22(4)	8(2)	8(3)	9(3)	8(2)	8(2)
φ_{38}	20	9(2)	20(4)	22(6)	17(6)	21(4)	8(2)
φ_{37}	5	5(2)	8(4)	9(4)	14(5)	9(4)	5(2)
$\overline{\varphi_{13}}$	8	8(2)	19(2)	8(2)	8(2)	8(2)	8(2)
$\overline{\varphi_{14}}$	7	7(2)	16(3)	7(3)	7(7)	7(3)	7(7)
φ_{35}	4	4(2)	4(4)	7(4)	9(5)	7(4)	4(2)
φ_{23}	3	3(2)	4(4)	7(6)	3(2)	7(4)	3(2)
$\overline{\varphi_{43}}$	5	5(2)	8(2)	5(2)	11(2)	5(2)	5(2)
$\overline{\varphi_{48}}$	5	5(2)	8(2)	5(2)	11(2)	5(2)	5(2)
$\overline{\varphi_{24}}$	4	8(3)	4(2)	4(2)	4(3)	4(2)	4(2)
$\overline{\varphi_{35}}$	4	8(4)	4(2)	4(3)	5(3)	4(2)	4(2)
φ_{24}	3	3(2)	6(6)	3(4)	5(4)	3(4)	3(2)
$\overline{\varphi_{37}}$	5	9(4)	5(2)	5(3)	6(3)	5(2)	5(2)
$\overline{\varphi_{40}}$	4	7(3)	4(2)	5(2)	6(4)	5(2)	4(2)
$\overline{\varphi_{28}}$	5	5(2)	8(2)	5(2)	6(2)	5(2)	5(2)
φ_{29}	3	3(2)	4(4)	4(4)	5(2)	4(4)	3(2)
$\overline{\varphi_{23}}$	4	4(2)	4(2)	4(2)	6(2)	4(2)	4(2)
φ_{40}	6	5(2)	4(3)	6(5)	5(3)	6(4)	4(2)
φ_{45}	6	4(2)	4(3)	5(5)	5(3)	5(4)	4(2)
$\overline{\varphi_{41}}$	4	4(2)	6(2)	4(2)	4(2)	4(2)	4(2)
$\overline{\varphi_{34}}$	4	4(2)	6(2)	6(2)	6(2)	6(2)	6(2)
$\overline{\varphi_{32}}$	4	4(2)	4(2)	4(2)	6(2)	4(2)	4(2)
$\overline{\varphi_{46}}$	4	4(2)	6(2)	4(2)	4(2)	4(2)	4(2)
$\overline{\varphi_{20}}$	2	3(3)	2(2)	2(2)	2(3)	2(2)	2(2)
φ_4	3	3(2)	4(4)	3(4)	3(2)	3(2)	3(2)
$\overline{\varphi_9}$	3	4(3)	3(2)	3(2)	3(3)	3(2)	3(2)
φ_{18}	3	3(2)	4(4)	4(4)	3(2)	4(2)	3(2)
$\overline{\varphi_{25}}$	3	3(2)	3(2)	3(2)	4(2)	3(2)	3(2)
φ_{25}	3	3(2)	3(2)	4(2)	3(2)	4(2)	3(2)
$\overline{\varphi_{22}}$	3	4(4)	3(2)	3(2)	4(3)	3(2)	3(2)
φ_{22}	3	3(2)	4(3)	3(4)	4(3)	3(4)	3(2)
φ_{42}	3	4(2)	4(3)	3(4)	4(3)	3(4)	4(2)
φ_{47}	3	4(2)	4(3)	3(4)	4(3)	3(4)	4(2)
φ_{17}	4	5(3)	4(2)	4(2)	5(3)	4(2)	4(3)
$\overline{\varphi_{26}}$	4	4(2)	5(2)	4(2)	4(2)	4(2)	4(2)
$\overline{\varphi_{31}}$	4	4(2)	5(2)	5(2)	5(2)	5(2)	5(2)
φ_{31}	4	4(2)	5(2)	5(2)	5(2)	5(2)	5(2)
$\overline{\varphi_{27}}$	4	4(2)	4(2)	4(2)	5(2)	4(2)	4(2)
φ_{27}	4	4(2)	4(2)	5(2)	4(2)	5(2)	4(2)
$\overline{\varphi_{42}}$	4	5(4)	4(2)	5(2)	4(2)	5(2)	5(2)
$\overline{\varphi_{47}}$	4	5(4)	4(2)	5(2)	4(2)	5(2)	5(2)
$\overline{\varphi_{29}}$	4	4(2)	4(2)	4(2)	5(2)	4(2)	4(2)
φ_{32}	4	4(2)	4(2)	5(2)	4(2)	5(2)	4(2)

Table 6 continued

LTL	N_1	N_2	D_1	D_2	LD_1	LD_2	LD_p
$\overline{\varphi_{33}}$	5	5(2)	6(2)	6(2)	6(2)	6(2)	6(2)
$\overline{\varphi_1}$	2	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)
φ_1	1	1(2)	1(2)	1(2)	1(2)	1(2)	1(2)
$\overline{\varphi_{15}}$	2	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)
φ_{15}	2	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)
$\overline{\varphi_2}$	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
φ_2	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
$\overline{\varphi_6}$	2	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)
φ_6	2	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)
$\overline{\varphi_{16}}$	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
φ_{16}	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
$\overline{\varphi_3}$	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
$\overline{\varphi_5}$	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)

Table 7 This table is a continuation of Table 6

LTL	N_1	N_2	D_1	D_2	LD_1	LD_2	LD_p
$\overline{\varphi_4}$	4	4(2)	4(2)	4(2)	4(2)	4(2)	4(2)
φ_{20}	2	2(2)	2(3)	2(4)	2(3)	2(4)	2(2)
φ_3	2	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)
φ_5	2	2(2)	2(4)	2(2)	2(2)	2(2)	2(2)
$\overline{\varphi_7}$	2	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)
$\overline{\varphi_{17}}$	3	3(2)	3(3)	3(4)	3(3)	3(4)	3(3)
φ_7	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
$\overline{\varphi_8}$	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
$\overline{\varphi_{19}}$	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
$\overline{\varphi_{18}}$	4	4(2)	4(2)	4(2)	4(2)	4(2)	4(2)
φ_9	2	2(2)	2(3)	2(4)	2(3)	2(4)	2(2)
φ_8	2	2(2)	2(4)	2(2)	2(2)	2(2)	2(2)
φ_{19}	2	2(2)	2(4)	2(2)	2(2)	2(2)	2(2)
$\overline{\varphi_{21}}$	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
$\overline{\varphi_{30}}$	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
φ_{21}	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
φ_{30}	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
$\overline{\varphi_{10}}$	6	6(2)	6(2)	6(2)	6(2)	6(2)	6(2)
φ_{10}	5	5(2)	5(2)	5(2)	5(2)	5(2)	5(2)
φ_{26}	4	4(2)	4(2)	4(2)	4(2)	4(2)	4(2)
$\overline{\varphi_{36}}$	5	5(2)	5(2)	5(2)	5(2)	5(2)	5(2)
φ_{41}	4	4(2)	4(2)	4(2)	4(2)	4(2)	4(2)
φ_{36}	6	6(2)	6(2)	6(2)	6(2)	6(2)	6(2)
$\overline{\varphi_{12}}$	7	7(2)	7(2)	7(2)	7(2)	7(2)	7(2)
φ_{12}	6	6(2)	6(2)	6(2)	6(2)	6(2)	6(2)
φ_{46}	4	4(2)	4(2)	4(2)	4(2)	4(2)	4(2)
$\overline{\varphi_{11}}$	7	7(2)	7(2)	7(2)	7(2)	7(2)	7(2)
φ_{11}	7	7(2)	7(2)	7(2)	7(2)	7(2)	7(2)

Table 8 This table is a continuation of Table 4

LTL	N_1	N_2	D_1	D_2	LD_1	LD_2	LD_p
$\overline{\chi_{8,3}}$	35(2)	179(4)	28(9)	57(5)	43(3)	57(3)	12(4)
$\chi_{9,2}$	22(3)	17(4)	4(6)	4(6)	4(6)	4(6)	2(4)
$\overline{\chi_{7,4}}$	5(2)	37(4)	4(3)	4(3)	4(3)	4(3)	4(3)
$\overline{\chi_{3,3}}$	11	17(3)	56(8)	13(2)	82(9)	13(2)	10(2)
$\chi_{7,2}$	9(2)	21(3)	2(4)	2(4)	2(4)	2(4)	2(3)
$\overline{\chi_{6,3}}$	3	23(3)	6(2)	6(2)	3(2)	3(2)	3(2)
$\overline{\chi_{11,2}}$	7	44(2)	15(3)	7(3)	7(3)	7(3)	7(2)
$\overline{\chi_{9,4}}$	65(5)	132(8)	24(6)	24(6)	24(6)	24(6)	24(6)
$\overline{\chi_{9,3}}$	16(4)	35(6)	6(5)	6(5)	6(5)	6(5)	6(5)
$\chi_{8,3}$	20(2)	64(3)	14(4)	21(4)	12(4)	21(4)	12(4)
$\overline{\chi_{8,2}}$	8(2)	30(4)	6(7)	14(5)	8(3)	14(3)	6(4)
$\chi_{8,2}$	7(2)	21(3)	4(4)	7(4)	6(4)	7(4)	6(4)
$\chi_{10,4}$	16	73(2)	65(3)	42(3)	65(3)	42(3)	31(2)
$\overline{\chi_{9,2}}$	5(3)	12(6)	2(4)	2(4)	2(4)	2(4)	2(4)
$\chi_{10,3}$	8	36(2)	25(3)	16(3)	25(3)	16(3)	15(2)
$\overline{\chi_{10,4}}$	16	39(3)	65(2)	31(2)	31(2)	31(2)	31(2)
$\chi_{10,2}$	4	17(2)	9(3)	6(3)	9(3)	6(3)	6(3)
$\chi_{11,2}$	7	21(3)	27(2)	27(2)	7(2)	7(2)	7(2)
$\overline{\chi_{7,3}}$	4(2)	12(4)	3(3)	3(3)	3(3)	3(3)	3(3)
$\overline{\chi_{10,3}}$	8	19(3)	25(2)	15(2)	15(2)	15(2)	15(2)
$\chi_{4,4}$	42	42(2)	42(2)	115(2)	63(2)	115(2)	42(2)
$\chi_{3,3}$	12	21(2)	15(3)	11(4)	12(3)	11(4)	9(2)
$\chi_{5,3}$	1	3(3)	1(2)	1(2)	1(2)	1(2)	1(2)
$\overline{\chi_{7,2}}$	3(2)	5(4)	2(3)	2(3)	2(3)	2(3)	2(3)
$\chi_{5,4}$	1	3(3)	1(2)	1(2)	1(2)	1(2)	1(2)
$\chi_{5,5}$	1	3(3)	1(2)	1(2)	1(2)	1(2)	1(2)
$\overline{\chi_{2,4}}$	5	5(2)	11(2)	5(2)	5(2)	5(2)	5(2)
$\overline{\chi_{10,2}}$	4	9(3)	9(2)	7(2)	7(2)	7(2)	6(3)
$\chi_{4,3}$	18	18(2)	18(2)	36(2)	24(2)	36(2)	18(2)
$\chi_{11,1}$	3	6(3)	4(2)	4(2)	3(2)	3(2)	3(2)
$\overline{\chi_{3,2}}$	5	8(3)	6(4)	5(2)	9(5)	5(2)	5(2)
$\overline{\chi_{4,4}}$	41	41(2)	41(2)	41(2)	66(2)	41(2)	41(2)
$\overline{\chi_{4,3}}$	17	17(2)	17(2)	17(2)	26(2)	17(2)	17(2)
$\overline{\chi_{5,3}}$	1	2(2)	1(3)	1(3)	1(3)	1(3)	1(2)
$\overline{\chi_{5,4}}$	1	2(2)	1(3)	1(3)	1(3)	1(3)	1(2)
$\overline{\chi_{5,5}}$	1	2(2)	1(3)	1(3)	1(3)	1(3)	1(2)
$\overline{\chi_{4,2}}$	7	7(2)	7(2)	7(2)	10(2)	7(2)	7(2)
$\chi_{4,2}$	8	8(2)	8(2)	11(2)	9(2)	11(2)	8(2)
$\chi_{6,3}$	3	4(2)	3(3)	3(3)	3(3)	3(3)	3(2)
$\overline{\chi_{3,1}}$	3	4(3)	3(2)	3(2)	3(3)	3(2)	3(2)
$\overline{\chi_{11,1}}$	3	4(2)	4(3)	3(3)	3(3)	3(3)	3(2)
$\chi_{6,4}$	4	5(2)	4(3)	4(3)	4(3)	4(3)	4(2)
$\overline{\chi_{2,3}}$	4	4(2)	5(2)	4(2)	4(2)	4(2)	4(2)
$\chi_{3,2}$	4	5(2)	5(3)	5(4)	4(3)	5(4)	4(2)
$\chi_{6,5}$	5	6(2)	5(3)	5(3)	5(3)	5(3)	5(2)
$\overline{\chi_{2,2}}$	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
$\overline{\chi_{1,2}}$	4	4	4(2)	4(2)	4(2)	4(2)	4(2)

Table 8 continued

LTL	N_1	N_2	D_1	D_2	LD_1	LD_2	LD_p
$\chi_{2,2}$	3	3(2)	3(2)	3(2)	3(2)	3(2)	3(2)
$\chi_{1,2}$	4	4(2)	4(2)	4(2)	4(2)	4(2)	4(2)
$\chi_{3,1}$	2	2(2)	2(3)	2(4)	2(3)	2(4)	2(2)
$\overline{\chi_{1,3}}$	8	8	8(2)	8(2)	8(2)	8(2)	8(2)
$\chi_{2,3}$	4	4(2)	4(2)	4(2)	4(2)	4(2)	4(2)
$\chi_{1,3}$	8	8(2)	8(2)	8(2)	8(2)	8(2)	8(2)
$\overline{\chi_{1,4}}$	16	16	16(2)	16(2)	16(2)	16(2)	16(2)
$\chi_{2,4}$	5	5(2)	5(2)	5(2)	5(2)	5(2)	5(2)
$\chi_{1,4}$	16	16(2)	16(2)	16(2)	16(2)	16(2)	16(2)

Table 9 This table displays the sizes of the intermediate automata for the 'Dwyer'-patterns set. The table list number of states, followed by the number of acceptance sets (if larger than 1) and is sorted descending in regards to the largest difference in order of magnitude differences, as explained in the text. We write $\frac{1}{n}\Sigma, \sigma, \sqrt[n]{\Pi}$, and med., for the average, the standard deviation, the geometric average, and the median, respectively, for the number of states. For D_1 and D_2 we list the sizes of the intermediate DRAs and for LD_1 and LD_2 the sizes of the intermediate LDBAs

LTL	D_1	D_2	LD_1	LD_2
φ_{49}	22(6)	118(16)	19	63
$\overline{\varphi_{14}}$	16(4)	7(2)	45	14
φ_{14}	11(6)	67(8)	14	47
φ_{44}	20(6)	75(14)	19	47
φ_{13}	22(4)	7(4)	14	29
$\overline{\varphi_{43}}$	8(2)	5(2)	19	5
$\overline{\varphi_{48}}$	8(2)	5(2)	19	5
φ_{39}	54(16)	123(8)	40	38
φ_{33}	20(8)	6(4)	14	13
φ_{43}	6(4)	11(6)	9	19
φ_{48}	6(4)	11(6)	9	19
φ_{23}	4(4)	6(6)	7	13
$\overline{\varphi_{39}}$	29(6)	10(4)	20	26
φ_{35}	4(6)	7(4)	11	8
φ_{24}	6(6)	3(4)	8	8
φ_{47}	4(2)	3(4)	7	8
φ_{28}	8(4)	15(6)	10	19
φ_{34}	8(12)	13(4)	19	15
$\overline{\varphi_{13}}$	19(2)	8(2)	8	8
$\overline{\varphi_{38}}$	8(2)	8(2)	18	9
φ_{4}	4(4)	3(4)	6	7
φ_{9}	2(2)	2(4)	4	5
$\overline{\varphi_{45}}$	5(8)	5(2)	11	5
φ_{42}	4(2)	3(4)	7	7
φ_{29}	4(4)	4(4)	9	8
$\overline{\varphi_{44}}$	18(14)	14(6)	27	20
$\overline{\varphi_{34}}$	6(2)	6(2)	12	6
φ_{18}	4(4)	4(4)	6	8

Table 9 continued

LTL	D_1	D_2	LD_1	LD_2
$\overline{\varphi_{23}}$	4(2)	4(2)	8	4
φ_{40}	4(2)	6(4)	8	6
φ_{45}	4(2)	5(4)	8	6
$\overline{\varphi_{32}}$	4(2)	4(2)	8	4
φ_{32}	4(2)	5(2)	8	6
$\overline{\varphi_{17}}$	4(2)	3(4)	5	6
$\overline{\varphi_{22}}$	3(2)	3(2)	6	4
φ_{22}	4(2)	3(4)	6	6
$\overline{\varphi_{40}}$	3(4)	5(2)	6	5
φ_{36}	6(2)	6(2)	11	7
$\overline{\varphi_{20}}$	2(2)	2(2)	4	3
φ_{20}	2(2)	2(4)	4	4
$\overline{\varphi_{28}}$	8(2)	5(2)	9	5
φ_{17}	4(2)	4(2)	7	6
$\overline{\varphi_{24}}$	4(2)	4(2)	7	5
$\overline{\varphi_{29}}$	4(2)	4(2)	7	4
φ_{37}	8(6)	9(4)	13	10
$\overline{\varphi_{49}}$	26(14)	19(6)	30	22
$\overline{\varphi_{9}}$	3(2)	3(2)	5	4
$\overline{\varphi_{25}}$	3(2)	3(2)	5	3
φ_{30}	3(2)	3(2)	5	4
φ_{31}	5(2)	5(2)	8	6
$\overline{\varphi_{35}}$	4(2)	4(2)	6	5
$\overline{\varphi_{27}}$	4(2)	4(2)	6	4
$\overline{\varphi_{41}}$	6(2)	4(2)	4	4
$\overline{\varphi_{46}}$	6(2)	4(2)	4	4
$\overline{\varphi_{37}}$	5(2)	5(2)	7	6
φ_{38}	20(4)	20(6)	27	22
φ_{2}	3(2)	3(2)	4	3
φ_{16}	3(2)	3(2)	4	3
φ_{7}	3(2)	3(2)	4	4
φ_{25}	3(2)	4(2)	4	4
φ_{21}	3(2)	3(2)	4	4
$\overline{\varphi_{26}}$	5(2)	4(2)	4	4
φ_{26}	4(2)	4(2)	5	4
φ_{27}	4(2)	5(2)	5	5
φ_{41}	4(2)	4(2)	5	5
$\overline{\varphi_{42}}$	4(2)	5(2)	5	5
$\overline{\varphi_{47}}$	4(2)	5(2)	5	5
φ_{46}	4(2)	4(2)	5	5
$\overline{\varphi_{11}}$	8(2)	7(2)	7	7
φ_{11}	7(2)	7(2)	8	7
$\overline{\varphi_{1}}$	2(2)	2(2)	2	2
φ_{1}	1(2)	1(2)	1	1
$\overline{\varphi_{15}}$	2(2)	2(2)	2	2
φ_{15}	2(2)	2(2)	2	2
$\overline{\varphi_{2}}$	3(2)	3(2)	3	3

Table 9 continued

LTL	D_1	D_2	LD_1	LD_2
$\overline{\varphi_{6}}$	2(2)	2(2)	2	2
φ_{6}	2(2)	2(2)	2	2
$\overline{\varphi_{16}}$	3(2)	3(2)	3	3
$\overline{\varphi_{3}}$	3(2)	3(2)	3	3
$\overline{\varphi_{5}}$	3(2)	3(2)	3	3
$\overline{\varphi_{4}}$	4(2)	4(2)	4	4
φ_{3}	2(2)	2(2)	2	2
φ_{5}	2(4)	2(2)	2	2
$\overline{\varphi_{7}}$	2(2)	2(2)	2	2
$\overline{\varphi_{8}}$	3(2)	3(2)	3	3
$\overline{\varphi_{19}}$	3(2)	3(2)	3	3
$\overline{\varphi_{18}}$	4(2)	4(2)	4	4
φ_{8}	2(4)	2(2)	2	2
φ_{19}	2(4)	2(2)	2	2
$\overline{\varphi_{21}}$	3(2)	3(2)	3	3
$\overline{\varphi_{30}}$	3(2)	3(2)	3	3
$\overline{\varphi_{10}}$	6(2)	6(2)	6	6
φ_{10}	5(2)	5(2)	5	5
$\overline{\varphi_{31}}$	5(2)	5(2)	5	5
$\overline{\varphi_{36}}$	5(2)	5(2)	5	5
$\overline{\varphi_{33}}$	6(2)	6(2)	6	6
$\overline{\varphi_{12}}$	7(2)	7(2)	7	7
φ_{12}	6(2)	6(2)	6	6
$\frac{1}{n}\Sigma$	6.52	8.68	8.21	8.29
σ	7.35	1.95	7.57	10.07
$\sqrt[n]{\Pi}$	4.75	4.80	6.16	5.68
med.	4.0	4.0	6.0	5.0

Table 10 This table displays the sizes of the intermediate automata for the parametrised set (Table 1). The table is structured as Table 9

LTL	D_1	D_2	LD_1	LD_2
$\overline{\chi_{8,4}}$	4(10)	201(4)	9	54
$\overline{\chi_{8,3}}$	3(8)	57(4)	7	26
$\chi_{8,4}$	5(10)	56(4)	10	21
$\overline{\chi_{8,2}}$	2(6)	14(4)	5	12
$\chi_{8,3}$	4(8)	21(4)	8	12
$\overline{\chi_{11,3}}$	62(2)	15(2)	16	16
$\chi_{7,4}$	1(10)	1(10)	6	6
$\overline{\chi_{6,5}}$	1(10)	1(10)	6	6
$\overline{\chi_{5,5}}$	1(2)	1(2)	6	6
$\chi_{9,4}$	96(16)	58(16)	29	29
$\overline{\chi_{3,3}}$	26(8)	13(2)	43	16
$\chi_{7,3}$	1(8)	1(8)	5	5
$\overline{\chi_{6,4}}$	1(8)	1(8)	5	5
$\overline{\chi_{5,4}}$	1(2)	1(2)	5	5
$\overline{\chi_{9,4}}$	1(8)	1(8)	5	5
$\chi_{4,4}$	42(2)	115(2)	67	112
$\overline{\chi_{3,2}}$	6(4)	5(2)	15	8
$\chi_{7,2}$	1(6)	1(6)	4	4
$\overline{\chi_{6,3}}$	1(6)	1(6)	4	4
$\overline{\chi_{5,3}}$	1(2)	1(2)	4	4
$\overline{\chi_{9,3}}$	1(6)	1(6)	4	4
$\chi_{9,2}$	2(6)	2(6)	6	6
$\overline{\chi_{10,4}}$	65(2)	31(2)	31	31
$\chi_{4,3}$	18(2)	36(2)	27	38
$\chi_{3,1}$	2(2)	2(4)	4	5
$\chi_{9,3}$	6(10)	10(10)	13	13
$\overline{\chi_{9,2}}$	1(4)	1(4)	3	3
$\overline{\chi_{2,4}}$	11(2)	5(2)	5	5
$\chi_{8,2}$	3(6)	7(4)	6	7
$\overline{\chi_{11,2}}$	15(2)	7(2)	8	8
$\chi_{10,4}$	65(2)	42(2)	33	33
$\chi_{3,3}$	15(2)	11(4)	20	22
$\chi_{3,2}$	5(2)	5(4)	8	10
$\chi_{11,1}$	2(4)	2(4)	4	4
$\overline{\chi_{10,3}}$	25(2)	15(2)	15	15
$\chi_{4,2}$	8(2)	11(2)	11	13
$\chi_{10,3}$	25(2)	16(2)	17	17
$\chi_{5,3}$	1(2)	1(2)	2	2
$\chi_{5,4}$	1(2)	1(2)	2	2
$\chi_{5,5}$	1(2)	1(2)	2	2
$\overline{\chi_{4,2}}$	7(2)	7(2)	11	7
$\overline{\chi_{3,1}}$	3(2)	3(2)	5	4
$\chi_{11,2}$	5(6)	5(6)	8	8
$\chi_{10,2}$	9(2)	6(2)	9	9
$\overline{\chi_{7,2}}$	2(2)	2(2)	3	3
$\overline{\chi_{4,3}}$	17(2)	17(2)	23	17
$\chi_{6,3}$	3(2)	3(2)	4	4
$\overline{\chi_{7,3}}$	3(2)	3(2)	4	4
$\overline{\chi_{11,1}}$	4(2)	3(2)	4	4
$\overline{\chi_{10,2}}$	9(2)	7(2)	7	7
$\chi_{6,4}$	4(2)	4(2)	5	5
$\overline{\chi_{7,4}}$	4(2)	4(2)	5	5
$\overline{\chi_{2,3}}$	5(2)	4(2)	4	4
$\overline{\chi_{4,4}}$	41(2)	41(2)	49	41
$\chi_{11,3}$	19(8)	19(8)	16	16
$\chi_{6,5}$	5(2)	5(2)	6	6
$\overline{\chi_{2,2}}$	3(2)	3(2)	3	3
$\overline{\chi_{1,2}}$	4(2)	4(2)	4	4
$\chi_{2,2}$	3(2)	3(2)	3	3
$\chi_{1,2}$	4(2)	4(2)	4	4
$\overline{\chi_{1,3}}$	8(2)	8(2)	8	8
$\chi_{2,3}$	4(2)	4(2)	4	4
$\chi_{1,3}$	8(2)	8(2)	8	8
$\overline{\chi_{1,4}}$	16(2)	16(2)	16	16
$\chi_{2,4}$	5(2)	5(2)	5	5
$\chi_{1,4}$	16(2)	16(2)	16	16
$\frac{1}{n}\Sigma$	11.26	14.79	10.82	12.29
σ	18.02	29.73	11.81	16.11
$\sqrt[n]{\Pi}$	4.85	5 .51	7.47	7.94
med.	4.0	5.0	6.0	6.0

Acknowledgements The authors want to thank Michael Luttenberger for helpful discussions and the anonymous reviewers for constructive feedback.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. T, Babiak, Křetínský, M., Rehák, V., Strejcek, J.: LTL to Büchi automata translation: Fast and more deterministic. In: TACAS. LNCS **7214**, 95–109 (2012). https://doi.org/10.1007/978-3-642-28756-5_8
2. Blahoudek, F., Heizmann, M., Schewe, S., Strejcek, J., Tsai, M.: Complementing semi-deterministic Büchi automata. In: TACAS. LNCS **9636**, 770–787 (2016). https://doi.org/10.1007/978-3-662-49674-9_49
3. Bloem, R., Chatterjee, K., Jobstmann, B.: Graph games and reactive synthesis. In: E.M. Clarke, T.A. Henzinger, H. Veith, R. Bloem (eds.) Handbook of Model Checking, pp. 921–962. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_27
4. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM **42**(4), 857–907 (1995). https://doi.org/10.1145/210332.210339
5. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 - A framework for LTL and ω-automata manipulation. In: C. Artho, A. Legay, D. Peled (eds.) Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings, *Lecture Notes in Computer Science*, vol. 9938, pp. 122–129 (2016). https://doi.org/10.1007/978-3-319-46520-3_8
6. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: M.A. Ardis, J.M. Atlee (eds.) Proceedings of the Second Workshop on Formal Methods in Software Practice, March 4-5, 1998, Clearwater Beach, Florida, USA, pp. 7–15. ACM (1998). https://doi.org/10.1145/298595.298598
7. Esparza, J., Křetínský, J.: From LTL to deterministic automata: a safraless compositional approach. In: CAV, *LNCS*, vol. 8559, pp. 192–208 (2014). https://doi.org/10.1007/978-3-319-08867-9_13
8. Esparza, J., Kretínský, J., Raskin, J., Sickert, S.: From LTL and limit-deterministic büchi automata to deterministic parity automata. In: A. Legay, T. Margaria (eds.) Tools and Algorithms for the Construction and Analysis of Systems—23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I, *Lecture Notes in Computer Science*, vol. 10205, pp. 426–442 (2017). https://doi.org/10.1007/978-3-662-54577-5_25
9. Esparza, J., Kretínský, J., Sickert, S.: One theorem to rule them all: A unified translation of LTL into ω-automata. In: A. Dawar, E. Grädel (eds.) Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018, pp. 384–393. ACM (2018). https://doi.org/10.1145/3209108.3209161
10. Esparza, J., Křetínský, J., Sickert, S.: From LTL to deterministic automata: a safraless compositional approach. Formal Methods in Syst. Des. **49**(3), 219–271 (2016). https://doi.org/10.1007/s10703-016-0259-2
11. Etessami, K., Holzmann, G.J.: Optimizing büchi automata. In: CONCUR, pp. 153–167 (2000). https://doi.org/10.1007/3-540-44618-4_13
12. Finkbeiner, B.: Automata, games, and verification . (2015)https://www.react.uni-saarland.de/teaching/automata-games-verification-15/downloads/notes.pdf
13. 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). https://doi.org/10.1016/j.ic.2014.12.021
14. Geldenhuys, J., Hansen, H.: Larger automata and less work for LTL model checking. In: SPIN. LNCS **3925**, 53–70 (2006). https://doi.org/10.1007/11691617_4
15. Jacobs, S., Basset, N., Bloem, R., Brenguier, R., Colange, M., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Michaud, T., Pérez, G.A., Raskin, J., Sankur, O., Tentrup, L.: The 4th reactive synthesis competition (SYNTCOMP 2017): Benchmarks, participants & results. In: D. Fisman, S. Jacobs (eds.) Proceedings Sixth Workshop on Synthesis, SYNT@CAV 2017, Heidelberg, Germany, 22nd July 2017, *EPTCS*, vol. 260, pp. 116–143 (2017). https://doi.org/10.4204/EPTCS.260.10
16. Jacobs, S., Bloem, R., Colange, M., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Luttenberger, M., Meyer, P.J., Michaud, T., Sakr, M., Sickert, S., Tentrup, L., Walker, A.: The 5th reactive synthesis competition (SYNTCOMP 2018): Benchmarks, participants & results. CoRR **abs/1904.07736** (2019)
17. Kähler, D., Wilke, T.: Complementation, disambiguation, and determinization of büchi automata unified. In: L. Aceto, I. Damgård, L.A. Goldberg, M.M. Halldórsson, A. Ingólfsdóttir, I. Walukiewicz (eds.) Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games, *Lecture Notes in Computer Science*, vol. 5125, pp. 724–735. Springer (2008). https://doi.org/10.1007/978-3-540-70575-8_59
18. Kini, D., Viswanathan, M.: Limit deterministic and probabilistic automata for LTL \ GU. In: TACAS, *LNCS*, vol. 9035, pp. 628–642 (2015). https://doi.org/10.1007/978-3-662-46681-0_57
19. Kretínský, J., Manta, A., Meggendorfer, T.: Semantic labelling and learning for parity game solving in LTL synthesis. In: ATVA, *Lecture Notes in Computer Science*, vol. 11781, pp. 404–422. Springer (2019)
20. Kretínský, J., Meggendorfer, T., Sickert, S.: Owl: A library for ω-words, automata, and LTL. In: S.K. Lahiri, C. Wang (eds.) Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings, *Lecture Notes in Computer Science*, vol. 11138, pp. 543–550. Springer (2018). https://doi.org/10.1007/978-3-030-01090-4_34
21. Kretínský, J., Meggendorfer, T., Waldmann, C., Weininger, M.: Index appearance record for transforming rabin automata into parity automata. In: A. Legay, T. Margaria (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I, *Lecture Notes in Computer Science*, vol. 10205, pp. 443–460 (2017). https://doi.org/10.1007/978-3-662-54577-5_26
22. Kupferman, O., Rosenberg, A.: The blowup in translating LTL to deterministic automata. In: MoChArt, *LNCS*, vol. 6572, pp. 85–94. Springer (2010)
23. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Trans. Comput. Log. **2**(3), 408–429 (2001). https://doi.org/10.1145/377978.377993
24. Löding, C.: Optimal bounds for transformations of omega-automata. In: C.P. Rangan, V. Raman, R. Ramanujam (eds.) Foundations of Software Technology and Theoretical Computer Science, 19th Conference, Chennai, India, December 13–15, 1999, Proceedings, *Lecture Notes in Computer Science*, vol. 1738, pp. 97–109. Springer (1999)

25. Löding, C., Pirogov, A.: Determinization of büchi automata: Unifying the approaches of safra and muller-schupp. In: C. Baier, I. Chatzigiannakis, P. Flocchini, S. Leonardi (eds.) 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9–12, 2019, Patras, Greece, *LIPIcs*, vol. 132, pp. 120:1–120:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.120
26. Löding, C., Pirogov, A.: New optimizations and heuristics for determinization of büchi automata. In: Y. Chen, C. Cheng, J. Esparza (eds.) Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28–31, 2019, Proceedings, *Lecture Notes in Computer Science*, vol. 11781, pp. 317–333. Springer (2019). https://doi.org/10.1007/978-3-030-31784-3_18
27. Luttenberger, M., Meyer, P.J., Sickert, S.: Practical synthesis of reactive systems from LTL specifications via parity games. Acta Inf (2019). https://doi.org/10.1007/s00236-019-00349-3
28. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit reactive synthesis strikes back! In: CAV (I), pp. 578–586 (2018). https://doi.org/10.1007/978-3-319-96145-3_31
29. Müller, D., Sickert, S.: LTL to deterministic emerson-lei automata. In: GandALF, pp. 180–194 (2017). https://doi.org/10.4204/EPTCS.256.13
30. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. Log. Methods Comput. Sci. (2007). https://doi.org/10.2168/LMCS-3(3:5)2007
31. Redziejowski, R.R.: An improved construction of deterministic omega-automaton using derivatives. Fundam. Inf. **119**(3–4), 393–406 (2012). https://doi.org/10.3233/FI-2012-744
32. Safra, S.: On the complexity of omega-automata. In: FOCS, pp. 319–327 (1988). https://doi.org/10.1109/SFCS.1988.21948
33. Sickert, S.: Linear temporal logic. Archive of Formal Proofs (2016). https://www.isa-afp.org/entries/LTL.shtml
34. Sickert, S.: A unified translation of linear temporal logic to ω-automata. Ph.D. thesis, Technical University of Munich, Germany (2019). http://nbn-resolving.de/urn:nbn:de:bvb:91-diss-20190801-1484932-1-4
35. Sickert, S., Esparza, J.: An efficient normalisation procedure for linear temporal logic and very weak alternating automata. In: LICS 2020 (under submission)
36. Sickert, S., Esparza, J., Jaax, S., Kretínský, J.: Limit-deterministic büchi automata for linear temporal logic. In: Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II, pp. 312–332 (2016). https://doi.org/10.1007/978-3-319-41540-6_17
37. Somenzi, F., Bloem, R.: Efficient büchi automata from LTL formulae. In: CAV, pp. 248–263 (2000). https://doi.org/10.1007/10722167_21
38. Tabakov, D., Rozier, K.Y., Vardi, M.Y.: Optimized temporal monitors for systemC. Formal Methods Syst. Des. **41**(3), 236–268 (2012). https://doi.org/10.1007/s10703-011-0139-8
39. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: FOCS, pp. 327–338 (1985). https://doi.org/10.1109/SFCS.1985.12
40. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: LICS, pp. 332–344 (1986)