Good-for-MDPs Automata

Ernst Moritz Hahn
School of EEECS, Queen's University Belfast, UK

Mateo Perez, Fabio Somenzi, Ashutosh Trivedi
University of Colorado Boulder, USA

Sven Schewe, Dominik Wojtczak
University of Liverpool, UK

Abstract

We characterize the class of nondeterministic ω-automata that can be used for the analysis of finite Markov decision processes (MDPs). We call these automata 'good-for-MDPs' (GFM). We show that GFM automata are closed under classic simulation as well as under more powerful simulation relations that leverage properties of optimal control strategies for MDPs. This closure enables us to exploit state-space reduction techniques, such as those based on direct and delayed simulation, that guarantee simulation equivalence. We demonstrate the promise of GFM automata by defining a new class of automata with favorable properties—they are Büchi automata with low branching degree obtained through a simple construction—and show that going beyond limit-deterministic automata may significantly benefit reinforcement learning.

Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases good for games omega automata, slim automata, limit determinism

1 Introduction

System specifications are often captured in the form of finite automata over infinite words (ω-automata), which are then used for model checking, synthesis, and learning. Of the commonly-used types of ω-automata, Büchi automata have the simplest acceptance condition, but require nondeterminism to recognize all ω-regular languages. Nondeterministic machines can use unbounded look-ahead to resolve nondeterministic choices. However, important applications—like reactive synthesis or model checking and reinforcement learning (RL) for Markov Decision Process (MDPs [23])—have a game setting, which requires the resolution of nondeterminism to be based on history.

Being forced to resolve nondeterminism on the fly, an automaton may end up rejecting words it should accept, so that using it can lead to incorrect results. Due to this difficulty, initial solutions to these problems have been based on deterministic automata—usually with Rabin or parity acceptance conditions. For two-player games, Henzinger and Piterman proposed the notion of *good-for-games* automata [14]. These are nondeterministic automata that simulate [20, 13, 8] a deterministic automaton that recognizes the same language. The existence of a simulation strategy means that nondeterministic choices can be resolved without look-ahead.

The situation is better in the case of probabilistic model checking, because the game for which a strategy is sought is played on an MDP against "blind nature," rather than against a strategic opponent who may take advantage of the automaton's inability to resolve nondeterminism on the fly. As early as 1985, Vardi noted that probabilistic model checking can be performed with Büchi automata endowed with a limited form of nondeterminism [34]. *Limit deterministic Büchi automata (LDBA)* [4, 11, 28] perform no nondeterministic choice after seeing an accepting transition. Still, they recognize all ω-regular languages and are, under mild restrictions [28], *suitable* for probabilistic model checking.

The stronger adversary assumed in two-player games makes good-for-games automata safe for probabilistic model checking, while *suitable limit-deterministic Büchi automata (SLDBAs)* are not generally safe for two-player games. The question then arises of how to identify the automata that may be used for probabilistic model checking, or, as we shall call them, are *good for MDPs (GFM)*.

We look for other types of nondeterministic Büchi automata (NBA) that are GFM besides the limit-deterministic ones. We start with the observation that the GFM property is preserved by simulation.

ArXiv 2019.

This gives access to simulation-based statespace reduction. As a side observation, we show that this also holds for good-for-games automata. Simulation-based statespace reduction is a powerful tool for Büchi automata, and, as we do now have Büchi automata to start with—the SLDBAs—we can apply the power of general simulation-based statespace reduction [6, 31, 10, 8]. This includes cheap quotienting (and thus simulation) techniques for the deterministic parts of SLDBAs [25].

In a second step, we consider a simulation game that takes the structure of finite MDPs and properties of games against "blind Nature" into account. This simulation exploits properties of accepting end-components (AECs); we thus refer to it as *AEC simulation*. This type of simulation does not appear to be suitable for statespace reduction; however, once a GFM automaton $\mathcal{G}$ is produced from a language equivalent automaton $\mathcal{A}$, it can be used to establish that $\mathcal{A}$ is GFM. This may allow us to use the small automaton $\mathcal{A}$ instead of the—usually much larger—GFM automaton $\mathcal{G}$ that was, e.g., produced by limit determinization.

Finally, we look at a recent application of Büchi automata in RL: In this context, the Büchi acceptance condition of SLDBAs [34] has proven to be more suitable than Rabin or parity conditions [12]. This observation extends to good-for-games Büchi automata, because they are no more expressive than deterministic Büchi automata (DBA) [30]. The drawback of using SLDBAs is that the transition from the initial to the accepting part often allows for a large number of choices. We use simulation to establish the GFM property of a simple breakpoint-based construction [21, 24] with a different form of limited nondeterminism: It never offers more than two successors. Experiments suggest that extending the class of GFM automata is beneficial to RL.

Related Work and Contribution. The production of deterministic and limit deterministic automata for model checking has been intensively studied [24, 22, 1, 26, 33, 32, 27, 28, 7, 29, 18], and several tools are available to produce different types of automata, including MoChiBA/Owl [28, 29, 18], LTL3BA [1], GOAL [33, 32], SPOT [7], Rabinizer [17], and Büchifier [15].

So far, only deterministic and a (slightly restricted [28]) class of limit deterministic automata have been considered for probabilistic model checking [34, 4, 11, 28]. Thus, while there have been advances in the efficient production of such automata [11, 28, 29, 18], the consideration of SLDBAs by Courcoubetis and Yannakakis in 1988 [3] has been the last time when a fundamental change in the automata foundation of MDP model checking has occurred.

The simple but effective observation that simulation preserves the suitability for MDPs—for both traditional simulation and the AEC simulation we introduce—extends the class of automata that can be used. The *slim automata* discussed in Section 3.2 are a first example of nondeterministic automata with appealing properties—they have a branching degree of at most 2 and a Büchi acceptance mechanism—that can now be used for MDP model checking and RL.

Another practical implication is the fact that one can now use standard minimization techniques on good-for-MDP automata that produce automata that simulate the source automata, such as state-space reduction techniques based on direct and delayed simulation [8]. While simulation-based techniques are powerful tools to reduce the statespace of an automaton, they do not preserve (limit) determinancy.

Moreover, AEC simulation may establish *a posteriori* that optimized nondeterministic automata that occur as intermediate stages in the construction of larger GFM automata are also GFM.

2 Preliminaries

An ω-*word* w on an alphabet Σ is a function $w : \mathbb{N} \to \Sigma$. We abbreviate $w(i)$ by w_i. The set of ω-words on Σ is written Σ^ω and a subset of Σ^ω is an ω-*language*.

Automata on Infinite Words. A *nondeterministic* ω-*automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, \mathsf{Acc} \rangle$, where Σ is a finite *alphabet*, Q is a finite set of *states*, $Q_0 \subseteq Q$ are the *initial states*, $\Delta \subseteq Q \times \Sigma \times Q$ are transitions, and Acc is the *acceptance condition*, to be discussed presently. A *run* r of $\mathcal{A}$ on $w \in \Sigma^\omega$ is an ω-word $r_0, w_0, r_1, w_1, \ldots$ in $(Q \times \Sigma)^\omega$ such that $r_0 \in Q_0$ and, for $i > 0$, it is $(r_{i-1}, w_{i-1}, r_i) \in \Delta$.

We write $\inf(r)$ for the set of transitions that appear infinitely often in the run r. It is sometimes useful to require that there is only one initial state. We then write $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, \mathsf{Acc} \rangle$.

We consider two types of acceptance conditions—Büchi and Rabin—that depend on the transitions that occur infinitely often in a run of an automaton. The *Büchi* acceptance condition defined by $\Gamma \subseteq Q \times \Sigma \times Q$ is the set of runs $\{r \in (Q \cup \Sigma)^\omega : \inf(r) \cap \Gamma \neq \emptyset\}$. A *Rabin* acceptance condition is defined in terms of k pairs of subsets of $Q \times \Sigma \times Q$, $(B_0, G_0), \ldots, (B_{k-1}, G_{k-1})$, as the set $\{r \in (Q \cup \Sigma)^\omega : \exists i < k \,.\, \inf(r) \cap B_i = \emptyset \wedge \inf(r) \cap G_i \neq \emptyset\}$. A *Büchi* (Rabin) automaton is an ω-automaton equipped with a Büchi (Rabin) acceptance condition.

A run r of $\mathcal{A}$ is *accepting* if $r \in \mathsf{Acc}$. The *language*, $L_\mathcal{A}$, of $\mathcal{A}$ (or, *recognized* by $\mathcal{A}$) is the subset of words in Σ^ω that have accepting runs in $\mathcal{A}$. A language is *ω-regular* if it is accepted by an ω-automaton. An automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, \mathsf{Acc} \rangle$ is *deterministic* if $(q, \sigma, q'), (q, \sigma, q'') \in \Delta$ implies $q' = q''$. $\mathcal{A}$ is *complete* if, for all $\sigma \in \Sigma$ and all $q \in Q$, there is a transition $(q, \sigma, q') \in \Delta$. A word in Σ^ω has exactly one run in a deterministic, complete automaton.

Markov Decision Processes. A *probability distribution* over a finite set S is a function $d : S \to [0, 1]$ such that $\sum_{s \in S} d(s) = 1$. Let $\mathcal{D}(S)$ denote the set of all discrete distributions over S. We say a distribution $d \in \mathcal{D}(S)$ is a *point distribution* if $d(s){=}1$ for some $s \in S$. For a distribution $d \in \mathcal{D}(S)$ we write $\mathrm{supp}(d) \stackrel{\text{def}}{=} \{s \in S : d(s) > 0\}$ for the support of d.

A *Markov decision process (MDP)* $\mathcal{M}$ is a tuple (S, A, T, Σ, L) where S is a finite set of states, A is a finite set of *actions*, $T : S \times A \rightharpoondown \mathcal{D}(S)$ is the *probabilistic transition function*, Σ is an alphabet, and $L : S \times A \times S \rightharpoondown \Sigma$ is the *labeling function* of the set of transitions. For any state $s \in S$, we let $A(s)$ denote the set of actions available in the state s. For states $s, s' \in S$ and $a \in A(s)$, we have that $T(s, a)(s')$ equals $\Pr(s'|s, a)$.

A *run* of $\mathcal{M}$ is an ω-word $\langle s_0, a_1, \ldots \rangle \in S \times (A \times S)^\omega$ such that $\Pr(s_{i+1}|s_i, a_{i+1}) > 0$ for all $i \geq 0$. A finite run is a finite such sequence. For a *run* $r = \langle s_0, a_1, s_1, \ldots \rangle$ we define corresponding labeled run as $L(r) = \langle L(s_0, a_1, s_1), L(s_1, a_2, s_2), \ldots \rangle \in \Sigma^\omega$. We write $\mathit{Runs}^\mathcal{M}$/$\mathit{Runs}_f^\mathcal{M}$ for the set of runs/finite runs of $\mathcal{M}$ and $\mathit{Runs}^\mathcal{M}(s)$/$\mathit{Runs_f}^\mathcal{M}(s)$ for the set of runs/finite runs of $\mathcal{M}$ starting from state s. For a finite run r we write $\mathrm{last}(r)$ for the last state of the sequence. A strategy in $\mathcal{M}$ is a function $\mu : \mathit{Runs}_f \to \mathcal{D}(A)$ such that $\mathrm{supp}(\mu(r)) \subseteq A(\mathrm{last}(r))$. Let $\mathit{Runs}_\mu^\mathcal{M}(s)$ denote the subset of runs $\mathit{Runs}^\mathcal{M}(s)$ that correspond to strategy μ with the initial state s. Let $\Sigma_\mathcal{M}$ be the set of all strategies. We say that a strategy μ is *pure* if $\mu(r)$ is a point distribution for all runs $r \in \mathit{Runs}_f^\mathcal{M}$ and we say that μ is *positional* if $\mathrm{last}(r) = \mathrm{last}(r')$ implies $\mu(r) = \mu(r')$ for all runs $r, r' \in \mathit{Runs}_f^\mathcal{M}$.

Probabilistic model checking. The behavior of an MDP $\mathcal{M}$ under a strategy μ with starting state s is defined on a probability space $(\mathit{Runs}_\mu^\mathcal{M}(s), \mathcal{F}_{\mathit{Runs}_\mu^\mathcal{M}(s)}, \Pr_s^\mu)$ over the set of infinite runs of μ from s. Given a random variable over the set of infinite runs $f{:}\mathit{Runs}^\mathcal{M} \to \mathbb{R}$, we write $\mathbb{E}_s^\mu\{f\}$ for the expectation of f over the runs of $\mathcal{M}$ from state s that follow strategy μ.

Given an MDP $\mathcal{M}$ and an ω-regular objective given as an ω-automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, \mathsf{Acc} \rangle$, we are interested in computing an optimal strategy satisfying the objective that the run of $\mathcal{M}$ is in the language of $\mathcal{A}$. We define the semantic satisfaction probability for $\mathcal{A}$ and a strategy μ from state s as:

$$\mathsf{PSemSat}_\mathcal{A}^\mathcal{M}(s,\mu) {=} \Pr{}_s^\mu\{r {\in} \mathit{Runs}_\mu^\mathcal{M}(s){:}L(r){\in}L_\mathcal{A}\} \text{ and } \mathsf{PSemSat}_\mathcal{A}^\mathcal{M}(s) {=} \sup_\mu \big(\mathsf{PSemSat}_\mathcal{A}^\mathcal{M}(s,\mu)\big).$$

When using automata for the analysis of MDPs, we need a syntactic variation of the acceptance condition. Given an MDP $\mathcal{M} = (S, A, T, \Sigma, L)$ with a designated initial state $s_0 \in S$, and a deterministic ω-automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, \mathsf{Acc} \rangle$, the *product* $\mathcal{M} \times \mathcal{A}$ is the tuple $(S \times Q, (s_0, q_0), A, T^\times, \mathsf{Acc}^\times)$. The probabilistic transition function $T^\times : (S \times Q) \times A \rightharpoondown \mathcal{D}(S \times Q)$ is defined by

$$T^\times((s,q),a)((s',q')) = \begin{cases} T(s,a)(s') & \text{if } (q, L(s,a,s'), q') \in \Delta \\ 0 & \text{otherwise.} \end{cases}$$

If $\mathcal{A}$ is a deterministic Büchi automaton (DBA), where Acc is defined by $\Gamma \subseteq Q \times \Sigma \times Q$, then $\Gamma^\times \subseteq (S \times Q) \times A \times (S \times Q)$ defines $\mathsf{Acc}^\times$ as follows: $((s,q),a,(s',q')) \in \Gamma^\times$ if, and only if, $(q, L(s,a,s'), q') \in \Gamma$ and $T(s,a)(s') \neq 0$. If $\mathcal{A}$ is a deterministic Rabin automaton (DRA) with k pairs, where $\mathsf{Acc} = \{(B_0,G_0),\ldots,(B_{k-1},G_{k-1})\}$, we define $\mathsf{Acc}^\times$ to be $\{(B_0^\times,G_0^\times),\ldots,(B_{k-1}^\times,G_{k-1}^\times)\}$ such that $((s,q),a,(s',q')) \in B_i^\times$ if and only if $(q,L(s,a,s'),q') \in B_i$ and $T(s,a)(s') \neq 0$; and likewise for $G_i^\times$. Likewise, a strategy μ on the MDP defines a strategy $\mu^\times$ on the product, and vice versa. We can define the syntactic satisfaction probability $\mathsf{PSynSat}_\mathcal{A}^\mathcal{M}((s,q),\mu^\times)$ as $\Pr_s^\mu\{r \in \mathit{Runs}_{\mu^\times}^{\mathcal{M}\times\mathcal{A}}((s,q)) : r \in \mathsf{Acc}^\times\}$ and $\mathsf{PSynSat}_\mathcal{A}^\mathcal{M}(s)$ as $\sup_{\mu^\times}\big(\mathsf{PSynSat}_\mathcal{A}^\mathcal{M}((s,q_0),\mu^\times)\big)$. Observe that $\mathsf{PSynSat}_\mathcal{A}^\mathcal{M}(s,\mu) = \mathsf{PSemSat}_\mathcal{A}^\mathcal{M}(s,\mu)$ trivially holds for a deterministic automaton $\mathcal{A}$.

Given an MDP $\mathcal{M} = (S,A,T,\Sigma,L)$, we define its underlying graph as the directed graph $\mathcal{G}_\mathcal{M} = (V,E)$ where $V = S$ and $E \subseteq S \times S$ is such that $(s,s') \in E$ if $T(s,a)(s') > 0$ for some $a \in A(s)$. A sub-MDP of $\mathcal{M}$ is an MDP $\mathcal{M}' = (S',A',T',\Sigma,L')$ where $S' \subset S$, $A' \subseteq A$ such that $A'(s) \subseteq A(s)$ for every $s \in S'$, and T' and L' are analogous to T and L when restricted to S' and A'. In particular, $\mathcal{M}'$ is closed under probabilistic transitions, i.e., for all $s \in S'$ and $a \in A'$ we have that $T(s,a)(s') > 0$ implies that $s' \in S'$. An *end-component* [5] of an MDP $\mathcal{M}$ is a sub-MDP $\mathcal{M}'$ of $\mathcal{M}$ such that $\mathcal{G}_{\mathcal{M}'}$ is strongly connected. A maximal end-component is an end-component that is maximal under set-inclusion. Every state of an MDP belongs to at most one maximal end-component.

▶ **Theorem 1** (End-Component Properties. Theorem 3.1 and Theorem 4.2 of [5]). *Once an end-component C of an MDP is entered, there is a strategy that visits every state-action combination in C infinitely often with probability 1 and stays in C forever.*

For an MDP with a Rabin condition $\mathsf{Acc} = \{(B_0,G_0),\ldots,(B_{k-1},G_{k-1})\}$*, an* accepting end-component *(AEC) is an end-component, for which there is an $i < k$ such that C contains a transition in G_i, but no transition in B_i. There is a positional pure strategy for an AEC C that surely stays in C and almost surely visits a transition in G_i infinitely often.*

For an MDP with some Rabin acceptance condition Acc *on its runs, there is a set of disjoint accepting end-components such that, from every state, the maximal probability to reach the union of these accepting end-components is the same as the maximal probability to satisfy the* Acc*. Moreover, this probability can be realized by combining a positional pure (reachability) strategy outside of this union with aforementioned positional pure strategies for the individual AECs.*

End-components and runs are defined for products just like for MDPs. A run of $\mathcal{M} \times \mathcal{A}$ is accepting if it satisfies the product's acceptance condition. An *accepting end-component* of $\mathcal{M} \times \mathcal{A}$ is an end-component such that every run of the product MDP that eventually dwells in it is accepting.

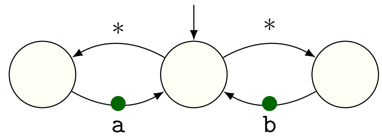
In view of Theorem 1, satisfaction of an ω-regular objective given by a deterministic automaton $\mathcal{A}$ by an MDP $\mathcal{M}$ can be formulated in terms of the accepting end-components of the product $\mathcal{M} \times \mathcal{A}$. The maximum probability of satisfaction for $\mathcal{A}$ by $\mathcal{M}$ is the maximum probability, over all strategies, that a run of the product $\mathcal{M} \times \mathcal{A}$ eventually dwells in one of its accepting end-components.

It is customary to use DRAs instead of DBAs in the construction of the product, because the latter cannot express all ω-regular objectives.

We make a similar definition for the product of MDPs and nondeterministic automata, with the difference that the set of actions is increased to also reflect the resolution of the nondeterministic choices made by the automaton [16, 12]. (Remark: This is the reason why using the slim automata that we define in Section 3.2 is attractive.)

For a nondeterministic automaton $\mathcal{A}$ and a language equivalent deterministic automaton $\mathcal{D}$, $\mathsf{PSynSat}_\mathcal{A}^\mathcal{M}(s,\mu) \leq \mathsf{PSemSat}_\mathcal{D}^\mathcal{M}(s,\mu)$ holds, but equality is not generally guaranteed, because the optimal resolution of nondeterministic choices may requires access to future events (see Figure 1).

Limit-Deterministic Büchi Automata. In spite of the large gap between the acceptance condition for DRAs and DBAs, even a very restricted form of nondeterminism is sufficient to make DBAs



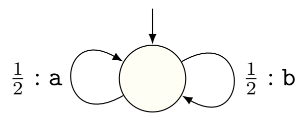


Figure 1 An NBA, which accepts all words over the alphabet $\{a, b\}$, that is not good for MDPs. The dotted transitions are accepting and $*$ stands for any letter of the alphabet. For the Markov chain on the right where the probability of a and b is $\frac{1}{2}$, the chance that the automaton makes infinitely many correct predictions is 0.

as expressive as DRAs. A *limit-deterministic* Büchi automaton (LDBA) is an nondeterministic Büchi automaton (NBA) $\mathcal{A} = \langle \Sigma, Q_i \cup Q_f, q_0, \Delta, \Gamma\rangle$ such that $Q_i \cap Q_f = \emptyset$; $\Gamma \subseteq Q_f \times \Sigma \times Q_f$; $(q, \sigma, q'), (q, \sigma, q'') \in \Delta$ and $q, q' \in Q_f$ implies $q' = q''$; and $(q, \sigma, q') \in \Delta$ and $q \in Q_f$ implies $q' \in Q_f$. Notice that an LDBA behaves deterministically once it has seen an accepting transition.

LDBAs are as expressive as general NBAs. Moreover, NBAs can be translated into LDBAs that can be used for the qualitative and quantitative analysis of MDPs [34, 4, 11, 28]. We use the translation from [11], which uses LDBAs that consist of two parts: an initial deterministic automaton (without accepting transitions) obtained by a subset construction; and a final part produced by a breakpoint construction. They are connected by a single "guess", where the automaton guesses a subset of the reachable states to start the breakpoint construction. Like other constructions (e.g. [28]), one can compose the resulting automaton with an MDP, such that the optimal control of the product defines a control on the MDP that maximizes the probability of obtaining a word from the language of the automaton. We refer to LDBAs with this property as *suitable* LDBAs (SLDBAs), cf. Theorem 5.

3 Good-for-MDP (GFM) Automata

We now discuss the construction and state-space reduction of automata that are *good for MDPs*.

▶ **Definition 2** (GFM automata)**.** *An ω-automaton $\mathcal{A}$ is* good for MDPs *if, for all MDPs $\mathcal{M}$, $\mathsf{PSynSat}^{\mathcal{M}}_{\mathcal{A}}(s_0) = \mathsf{PSemSat}^{\mathcal{M}}_{\mathcal{A}}(s_0)$ holds, where s_0 is the initial state of $\mathcal{M}$.*

For an automaton to match $\mathsf{PSemSat}^{\mathcal{M}}_{\mathcal{A}}(s_0)$, its nondeterminism is restricted, as it may not rely heavily on the future; it rather needs to be possible to resolve the nondeterminism on-the-fly. For example, the Büchi automaton presented on the left of Figure 1, which has to guess whether the next symbol is a or b, is not good for MDPs, because the simple Markov chain on the right of Figure 1 does not allow resolution of its nondeterminism on-the-fly.

There are three classes of automata that are known to be good for MDPs: (1) deterministic automata, (2) good for games automata [14, 16], and (3) limit deterministic automata that satisfy a few side constraints [4, 11, 28]. Especially in the context of RL, techniques that build on (3) are particularly useful, because these automata can use the simple Büchi acceptance condition for all ω-regular properties, which in turn can be translated to reachability goals [12]. Good for games (and deterministic) automata require more complex acceptance conditions—like parity—that are less suitable for RL, as they do not have a natural translation into rewards [12].

Using SLDBA [4, 11, 28] has the drawback that they would naturally have a high branching degree in the initial part, as they would naturally allow for many different transitions to the accepting part of the LDBA. This can in principle be avoided, but to the cost of a blow-up and a more complex construction and data structure [28]. We therefore suggest an alternative automata construction that produces nondeterministic Büchi automata with a small branching degree—it never produces more than two successors. We call these automata *slim*. The resulting automata are not (normally) limit deterministic, but we show that they are good for MDPs.

Due to technical dependencies we start with presenting a second observation, namely that automata that *simulate* language equivalent GFM automata are GFM. As a side result, we observe that the same holds for good-for-games automata. The side result is not surprising, as good-for-games automata were defined through simulation of deterministic automata [14]. But, to the best of our knowledge, the observation from Corollary 4 has not been made yet for good-for-games automata.

3.1 Simulating GFM

An automaton $\mathcal{A}$ *simulates* an automaton $\mathcal{B}$ if the duplicator wins the *simulation game*. The simulation game is played between a duplicator and a spoiler, who each control a pebble, which they move along the edges of $\mathcal{A}$ and $\mathcal{B}$, respectively. The game is started by the spoiler, who places her pebble on an initial state of $\mathcal{B}$. Next, the duplicator puts his pebble on an initial state of $\mathcal{A}$. The two players then take turns, always starting with the spoiler choosing an input letter and a transition for that letter in $\mathcal{B}$, followed by the duplicator choosing a transition for the same letter in $\mathcal{A}$. This way, both players produce an infinite run of their respective automaton. The duplicator has two ways to win a play of the game: if the run of $\mathcal{A}$ he constructs is accepting, and if the run the spoiler constructs on $\mathcal{B}$ is rejecting. The duplicator wins this game if he has a winning strategy, i.e., a recipe to move his pebble that guarantees that he wins. Such a winning strategy is "good-for-games," as it can only rely on the past. It can be used to transform winning strategies of $\mathcal{B}$, so that, if they were witnessing a good for games property or were good for an MDP, then the resulting strategy for $\mathcal{A}$ has the same property.

▶ **Lemma 3** (Simulation Properties)**.** *For ω-automata $\mathcal{A}$ and $\mathcal{B}$ the following holds.*

1. *If $\mathcal{A}$ simulates $\mathcal{B}$ then $\mathcal{L}(\mathcal{A}) \supseteq \mathcal{L}(\mathcal{B})$.*
2. *If $\mathcal{A}$ simulates $\mathcal{B}$ and $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*
3. *If $\mathcal{A}$ simulates $\mathcal{B}$, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$, and $\mathcal{B}$ is GFG, then $\mathcal{A}$ is GFG.*
4. *If $\mathcal{A}$ simulates $\mathcal{B}$, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$, and $\mathcal{B}$ is GFM, then $\mathcal{A}$ is GFM.*

Proof. Facts (1) and (2) are both well known and trivial observations. Fact (1) follows from the fact that an accepting run of $\mathcal{B}$ on a word α can be translated into an accepting run of $\mathcal{A}$ on α by using the winning strategy of $\mathcal{A}$ in the simulation game. Fact (2) follows immediately from Fact (1). Facts (3) and (4) follow by simulating the behaviour of $\mathcal{B}$ on each run. ◀

This observation allows us to use a family of state-space reduction techniques, in particular those based on language preserving translations for Büchi automata based on simulation relation [6, 31, 10, 8]. This requires stronger notions of simulations, like direct and delayed simulation [8]. For the deterministic part of LDBA, one can also use space reduction techniques for DBA like [25].

▶ **Corollary 4.** *All statespace reduction techniques that turn a nondeterministic automaton $\mathcal{A}$ into a nondeterministic automata $\mathcal{B}$ that simulates $\mathcal{A}$ preserve GFG and GFM: if $\mathcal{A}$ is GFG or GFM, then $\mathcal{B}$ is GFG or GFM, respectively.*

3.2 Constructing Slim GFM Automata

Let us fix Büchi automaton $\mathcal{B} = \langle \Sigma, Q, Q_0, \Delta, \Gamma \rangle$. We can write Δ as a function $\hat{\delta} \colon Q \times \Sigma \to 2^Q$ with $\hat{\delta} \colon (q, \sigma) \mapsto \{q' \in Q \mid (q, \sigma, q') \in \Delta\}$, which can be lifted to sets, using the deterministic transition function $\delta \colon 2^Q \times \Sigma \to 2^Q$ with $\delta \colon (S, \sigma) \mapsto \bigcup_{q \in S} \hat{\delta}(q, \sigma)$. We also define an operator, ndet, that translates deterministic transition functions $\delta \colon R \times \Sigma \to R$ to relations, using

$$\mathsf{ndet} \colon (R \times \Sigma \to R) \to 2^{R \times \Sigma \times R} \quad \text{with} \quad \mathsf{ndet} \colon \delta \mapsto \big\{(q, \sigma, q') \mid q' \in \delta(\{q\}, \sigma)\big\}.$$

This is just an easy means to move back and forth between functions and relations, and helps one to visualize the maximal number of successors. We next define the variations of subset and breakpoint constructions that are used to define the well-known limit deterministic GFM automata—which we use in our proofs—and the slim GFM automata we construct.

Let $3^Q := \{(S,S') \mid S' \subsetneq S \subseteq Q\}$ and $3^Q_+ := \{(S,S') \mid S' \subseteq S \subseteq Q\}$. We define the subset notation for the transitions and accepting transitions as $\delta_S, \gamma_S \colon 2^Q \times \Sigma \to 2^Q$ with

$$\delta_S \colon (S,\sigma) \mapsto \{q' \in Q \mid \exists q \in S.\, (q,\sigma,q') \in \Delta\} \text{ and } \gamma_S \colon (S,\sigma) \mapsto \{q' \in Q \mid \exists q \in S.\, (q,\sigma,q') \in \Gamma\}.$$

We define raw breakpoint transitions $\delta_R \colon 3^Q \times \Sigma \to 3^Q_+$ as $\big((S,S'),\sigma\big) \mapsto \big(\delta_S(S,\sigma), \delta_S(S',\sigma) \cup \gamma_S(S,\sigma)\big)$. In this construction, we follow the set of reachable states (first set) and the states that are reachable while passing at least one of the accepting transitions (second set).

To turn this into a breakpoint automaton, we reset the second set to the empty set when it equals the first; the transitions where we reset the second set are exactly the accepting ones. Formally, the breakpoint automaton $\mathcal{D} = \langle \Sigma, 3^Q, (Q_0, \emptyset), \delta_B, \gamma_B \rangle$ is defined such that, when $\delta_R \colon \big((S,S'),\sigma\big) \mapsto (R,R')$, then there are three cases:

1. if $R = \emptyset$, then $\delta_B\big((S,S')\big)$ is undefined (or, if a complete automaton is preferred, maps to a rejecting sink),
2. otherwise, if $R \neq R'$, then $\delta_B \colon \big((S,S'),\sigma\big) \mapsto (R,R')$ is a non-accepting transition,
3. otherwise $\delta_B, \gamma_B \colon \big((S,S'),\sigma\big) \mapsto (R',\emptyset)$ is an accepting transition.

Finally, we define transitions $\Delta_{SB} \subseteq 2^Q \times \Sigma \times 3^Q$ that lead from a subset to a breakpoint construction, and $\delta_{2,1} \colon 3^Q \times \Sigma \to 3^Q$ that promote the second set of a breakpoint construction to the first set as follows.

1. $\Delta_{SB} = \Big\{\big(S,\sigma,(S',\emptyset)\big) \mid \emptyset \neq S' \subseteq \delta_S(S,\sigma)\Big\}$ are non-accepting transitions,
2. if $\delta_S(S',\sigma) = \gamma_S(S,\sigma) = \emptyset$, then $\delta_{2,1}\big((S,S'),\sigma\big)$ is undefined, and
3. otherwise $\delta_{2,1} \colon \big((S,S'),\sigma\big) \mapsto \big(\delta_S(S',\sigma) \cup \gamma_S(S,\sigma), \emptyset\big)$ is an accepting transition.

We can now define standard limit deterministic good for MDP automata.

▶ **Theorem 5.** *[11] $\mathcal{A} = \langle \Sigma, 2^Q \cup 3^Q, Q_0, \mathsf{ndet}(\delta_S) \cup \Delta_{SB} \cup \mathsf{ndet}(\delta_B), \mathsf{ndet}(\gamma_B) \rangle$ recognizes the same language as $\mathcal{B}$. It is limit deterministic and good for MDPs.*

We now show how to construct a slim GFM Büchi automaton.

▶ **Theorem 6** (Slim GFM Büchi Automaton)**.** *The automaton*

$$\mathcal{S} = \langle \Sigma, 3^Q, (Q_0, \emptyset), \mathsf{ndet}(\delta_B) \cup \mathsf{ndet}(\delta_{2,1}), \mathsf{ndet}(\gamma_B) \cup \mathsf{ndet}(\delta_{2,1}) \rangle$$

simulates $\mathcal{A}$. $\mathcal{S}$ is slim, language equivalent to $\mathcal{B}$, and good for MDPs.

The most trivial observation is that $\mathcal{S}$ is slim: its set of transitions is the union of two sets of deterministic transitions. The proof idea for showing that $\mathcal{S}$ simulates $\mathcal{A}$ is to use transitions from $\delta_{2,1}$ when $\mathcal{S}$ is in the breakpoint part, in a way that guarantees that the first set of state of the pair of $\mathcal{S}$ always contains the first set of state of the pair of $\mathcal{A}$. A detailed proof is in Appendix A.1.

The resulting automata are simple in structure and enable symbolic implementation (See Fig. 2). It cannot be expected that there are much smaller good for MDP automata, as its explicit construction is the only non-polynomial part in model checking MDPs.

▶ **Theorem 7.** *Constructing a GFM Büchi automaton G that recognizes the models of an LTL formula φ requires time doubly exponential in φ, and constructing a GFM Büchi automaton G that recognizes the language of an NBA $\mathcal{B}$ requires time exponential in $\mathcal{B}$.*

Proof. As resulting automata are GFM, they can be used to model check MDPs $\mathcal{M}$ against this property, with cost polynomial in product of $\mathcal{M}$ and $\mathcal{G}$. If $\mathcal{G}$ could be produced faster (and if they could, consequently be smaller) than claimed, it will contradict the 2-EXPTIME- and EXPTIME-hardness [4] of these model checking problems. ◀

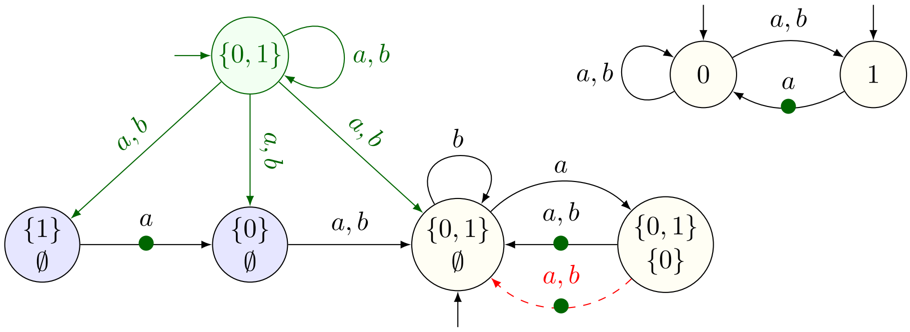


Figure 2 An NBA for G F a (in the upper right corner) together with an SLDBA and a slim NBA constructed from it. The SLDBA and the slim NBA are shown sharing their common part. State $\{0, 1\}$, produced by the subset construction, is the initial state of the SLDBA, while state $(\{0, 1\}, \emptyset)$—the initial state of the breakpoint construction—is the initial state of the slim NBA. States $(\{1\}, \emptyset)$ and $(\{0\}, \emptyset)$ are states of the breakpoint construction that only belong to the SLDBA because they are not reachable from $(\{0, 1\}, \emptyset)$. The transitions out of $\{0, 1\}$, except the self loop, belong to Δ_{SB}. The dashed-line transition from $(\{0, 1\}, \{0\})$ belongs to $\delta_{2,1}$

3.3 Accepting End-Component Simulation

Lemma 3 shows that the good for MDP property is preserved by simulation: For two language-equivalent automata $\mathcal{A}$ and $\mathcal{B}$, if $\mathcal{A}$ simulates $\mathcal{B}$ and $\mathcal{B}$ is GFM, then $\mathcal{A}$ is also GFM. However, a GFM automaton may not simulate a language-equivalent GFM automaton. (See Figure 3.) Therefore we introduce a coarser notion of simulation, Accepting End-Component (AEC) simulation, that takes advantage of the finiteness of the MDP $\mathcal{M}$. In particular, we can rely on Theorem 1 to focus on positional pure strategies for $\mathcal{M} \times \mathcal{B}$. Under such strategies, $\mathcal{M} \times \mathcal{B}$ becomes a Markov chain [2] such that almost all its runs have the following properties:

- They will eventually reach a leaf strongly connected component (LSCC) in the Markov chain.
- If they have reached a LSCC L, then, for all $\ell \in \mathbb{N}$, all sequences of transitions of length ℓ in L occur infinitely often, and no other sequence of length ℓ will occur.

With this in mind, we can intuitively ask the spoiler to pick a run through this Markov chain, and to disclose information about this run. Specifically, we can ask her to signal when she has reached an accepting LSCC[1] in the Markov chain, and to provide information about this LSCC, in particular information entailed by the full list of sequences of transitions of some fixed length ℓ described above. Runs that can be identified to either not reach an accepting LSCC, to visit transitions not in this list, or to visit only a subset of sequences from this list, form a 0 set. In the simulation game we define below, we try to make use of this observation and discard such runs.

A simulation game can only use the syntactic material of the automata—-neither the MDP nor the strategy are available. The information the spoiler may provide cannot explicitly refer to them. What we can ask the spoiler to provide is information on when she has entered an accepting LSCC, and, once she has signaled this, which sequences of length l of *automata* transitions of $\mathcal{B}$ occur in the LSCC. The set of sequences of automata transitions is simply the set of projections on the automata transitions from the set of sequences of transitions of length ℓ that occur in the LSCC L. We call this information a *gold-brim accepting end-component claim* of length ℓ, ℓ-GAEC claim for short.

[1] Note that there is nothing to show when a non-accepting LSCC is reached—if $\mathcal{B}$ rejects, then $\mathcal{A}$ may reject, too—and that there is nothing to show if no LSCC is reached, as this happens with probability 0.

The term "gold-brim" in the definition is used to indicate that this is a powerful approach, but not one that can be implemented efficiently. We will define weaker, efficiently implementable notions of accepting end-component claims (AEC claims) later.

The AEC simulation game is very similar to the simulation game of Section 3.1. Both players produce an infinite run of their respective automata. If the spoiler makes an AEC claim, e.g., an ℓ-GAEC claim, we say that her run *complies* with it if, starting with the transition when the AEC claim is made, all states, transitions, or sequences of transitions in the claim appear infinitely often, and all states, transitions, and sequences of transitions the claim excludes do not appear. For an ℓ-GAEC claim, this means that all of the sequences of transitions of length ℓ in the claim occur infinitely often, and no other sequence of length ℓ occurs henceforth.

Thus, like a classic simulation game, an ℓ-GAEC simulation game is started by the spoiler, who places her pebble on an initial state of $\mathcal{B}$. Next, the duplicator puts his pebble on an initial state of $\mathcal{A}$.

The two players then take turns, always starting with the spoiler choosing an input letter and an according transition from $\mathcal{B}$, followed by the duplicator choosing a transition for the same letter in $\mathcal{A}$.

Different from the classic simulation game, in an ℓ-GAEC simulation game, the spoiler has an additional move that she can (and, in order to win, has to) perform once in the game: In addition to choosing a letter and a transition, she can claim that she has reached an accepting end component, and provide a complete list of sequences of automata transitions of length ℓ that can henceforth occur. This store is maintained, and never updated. It has no further effect on the rules of the game: Both players produce an infinite run of their respective automata. The duplicator has four ways to win:

1. if the spoiler never makes an AEC claim,
2. if the run of $\mathcal{A}$ he constructs is accepting,
3. if the run the spoiler constructs on $\mathcal{B}$ does not comply with the AEC claim, and
4. if the run that the spoiler produces is not accepting.

For ℓ-GAEC claims, (4) simply means that the set of transitions defined by the sequences does not satisfy the Büchi, parity, or Rabin condition.

▶ **Theorem 8.** *[ℓ-GAEC Simulation] If $\mathcal{A}$ and $\mathcal{B}$ are language equivalent automata, $\mathcal{B}$ is good for MDPs, and there exists an ℓ such that $\mathcal{A}$ ℓ-GAEC simulates $\mathcal{B}$, then $\mathcal{A}$ is good for MDPs.*

For the proof, we use an arbitrary (but fixed) MDP $\mathcal{M}$, and an arbitrary (but fixed) pure optimal positional strategy μ for $\mathcal{M} \times \mathcal{B}$, resulting in the Markov chain $(\mathcal{M} \times \mathcal{B})_\mu$. We assume w.l.o.g. that the accepting LSCCs in $(\mathcal{M} \times \mathcal{B})_\mu$ are identified, e.g., by a bit.

Let τ be a winning strategy of the duplicator against the spoiler in an ℓ-GAEC simulation game. Abusing notation, we let $\tau \circ \mu$ denote the finite-memory strategyobtained from μ and τ for $\mathcal{M} \times \mathcal{A}$, where τ is acting only on the automata part of $(\mathcal{M} \times \mathcal{B})$, and where the spoiler makes the move to the end-component when she is in some LSCC B of $(\mathcal{M} \times \mathcal{B})_\mu$ and gives the full list of sequences of transitions of length ℓ that occur in B. The full proof is provided in Appendix A.2.

An ℓ-GAEC simulation, especially for large ℓ, results in very large state spaces, because the spoiler has to list all sequences of transitions of $\mathcal{B}$ of length ℓ that will appear infinitely often. No other sequence of length ℓ may then appear in the run[2]. This would, of course, be prohibitively expensive.

As a compromise, one can use coarser-grained information at the cost of reducing the duplicator's ability of winning the game. E.g., the spoiler could be asked to only reveal a transition that is repeated

[2] The AEC claim provides information about the accepting LSCC in the product automaton under the chosen pure positional strategy. When the AEC claim requires the exclusion of states, transitions, or sequences of transitions, then they are therefore surely excluded, whereas when it requires inclusion of, and thus inclusion of infinitely many occurrances of, states, trasitions, or sequences of transitions, then they (only) occur almost surely infinitely often. Yet, runs that do not contain them all infinitely often form a zero set, and can thus be ignored.

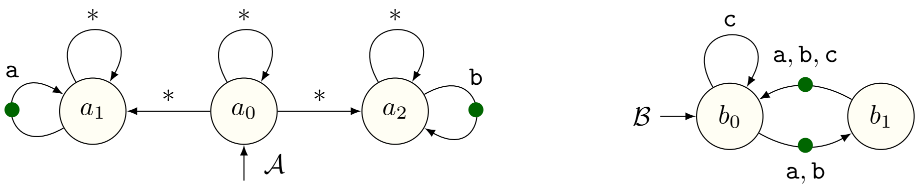


Figure 3 Automata $\mathcal{A}$ (left) and $\mathcal{B}$ (right) for $\varphi = (\mathsf{G}\,\mathsf{F}\,\mathtt{a}) \vee (\mathsf{G}\,\mathsf{F}\,\mathtt{b})$. The dotted transitions are accepting and $\star$ stands for any letter of the alphabet. The NBA $\mathcal{A}$ does not simulate the DBA $\mathcal{B}$: $\mathcal{B}$ can play a's until $\mathcal{A}$ moves to either the state on the left, or the state on the right. $\mathcal{B}$ can then easily win by henceforth playing only b's or only a's. However, $\mathcal{A}$ is good for MDPs. It can win the AEC simulation game by waiting until an AEC is reached (by $\mathcal{B}$), and then check if a or b occurs infinitely often in this AEC. Based on this knowledge, $\mathcal{A}$ can make its decision. This can be shown by AEC simulation if $\mathcal{B}$ has to provide sufficient information, such as a list of transitions—or even a list of letters—that occur infinitely often. The amount of information the spoiler has to provide therefore determines the strength of the AEC simulation used. If, e.g., $\mathcal{B}$ only has to reveal one accepting transition of the end-component, then it could select an endcomponent where the revealed transition is (b_1, c, b_0), which does not provide sufficient information. Whereas, if the duplicator is allowed to update the transition, then the duplicator wins by updating the recorded transition to the next a or b transition.

infinitely often, plus (when using more powerful acceptance conditions than Büchi), some acceptance information, say the dominating priority in a parity game or a winning Rabin pair. This type of coarse-grained claim can be refined slightly by allowing the duplicator to change the transition that is to appear infinitely often at any time to the transition just used by the spoiler.

Generally, we say that an AEC simulation game is any simulation game, where

- the spoiler provides a list of states, transitions, or sequences of transitions that will occur infinitely often and a list of states, transitions, or sequences of transitions that will not occur in the future when making her AEC claim, and
- the duplicator may be able to update this list based on his observations,
- there exists some ℓ-GAEC simulation game such that a winning strategy of the spoiler translates into a winning strategy of the spoiler in the AEC simulation game.

The requirement that a winning strategy of the spoiler translates into a winning spoiler strategy in an ℓ-GAEC game entails that AEC simulation games can establish the good for MDPs property.

▶ **Corollary 9.** *[AEC Simulation] If $\mathcal{A}$ and $\mathcal{B}$ are language equivalent automata, $\mathcal{B}$ is good for MDPs, and $\mathcal{A}$ AEC simulates $\mathcal{B}$, then $\mathcal{A}$ is good for MDPs.*

Of course, for every AEC simulation, one first has to establish the propety that winning strategies for the spoiler translate. We have used two simple variations of the AEC simulation games:

accepting transition: the spoiler may only make her AEC claim when taking an accepting transition; this transition—and no other information—is stored, and the spoiler commits to—and commits only to—seeing this transition infinitely often;

accepting transition with update: different to the *accepting transition* AEC simulation game, the duplicator can—but does not have to—update the stored accepting transition whenever the spoiler passes by an accepting transition.

▶ **Theorem 10.** *Both, the* accepted transition *and the* accepted transition with update *AEC simulation, can be used to establish the good for MDPs property.*

To show this (in Appendix A.3), we describe the strategy translations in accordance with Corollary 9.

We use AEC simulation to identify GFM automata among the automata produced (e.g., by SPOT [7]) at the beginning of the transformation. Figure 3 shows an example of an automaton, for which the duplicator wins the AEC simulation game, but loses the ordinary simulation game. Candidates for automata to simulate are, e.g., the slim GFM Büchi automata and the limit deterministic Büchi automata discussed above.

4 Evaluation

4.1 Random LTL formulas

As discussed, automata that simulate slim automata or limit deterministic automata are suitable for MDPs. This fact can be used to allow Büchi automata produced from general-purpose tools such as SPOT's [7] ltl2tgba rather than directly using specialized automata types. This has the advantage that automata produced by such tools are often smaller, as such general-purpose tools are highly optimized and because they are not restricted to producing slim or limit deterministic automata. Thus, what one has to do is to produce an arbitrary Büchi automaton using any available method, then transform this automaton into a slim or limit deterministic automaton, and then check whether the original automaton simulates the generated one.

We have evaluated this idea on a number of random LTL formulas. To do so, we have used SPOT's tool randltl. We have set the tree size, which influences the size of the formulas produced, to 50, and have produced 1000 formulas with 4 atomic propositions each. We left the other values to their defaults. We have then used SPOT's ltl2tgba (version 2.7) to turn these formulas into non-generalized Büchi automata using default options. Afterwards, for each automaton generated, we have used our tool to check whether the automaton simulates a limit deterministic automaton that we produce from this automaton. For comparison, we have also used Owl's [28] (version 18.06) tool ltl2ldba to compute limit deterministic non-generalized Buchi automata. We have also used the option of this tool to compute Büchi automata with a nondeterministic initial part. We used 10 minute timeouts.

From these 1000 formulas, 315 of the formulas can be transformed to deterministic Büchi automata. For an additional 492 other automata generated, standard simulation sufficed to show that they are suitable for MDPs. For a further 82 of them, the simplest AEC simulation (the spoiler chooses an accepting transition to occur infinitely often) sufficed, and another 3 of them could be classed GFM when strengthening AEC simulation by allowing the duplicator to update this transition. Only 36 automata turned out to be nonsimulatable and for 72 we did not get a decision (timeout).

For the LTL formulas, for which ltl2tgba could not produce deterministic automata, but for which simulation could be shown, the number of states in the generated automata was mostly lower than the number of states in the automata produced by Owl's tools. On average, the number of states per automaton was $\approx$15.04 for SPOT's ltl2tgba; while for Owl's ltl2ldba it was $\approx$68.84 (or $\approx$50.56, when the option to create a nondeterministic initial part is used).

Due to space restrictions, we cannot provide detailed information about all formulas and automata generated. Therefore, we provide examples for each outcome in Table 1 in Appendix B.1.

4.2 GFM Automata and Reinforcement Learning

SLDBAs have been used in [12] for model-free reinforcement learning of ω-regular objectives. While the Büchi acceptance condition allows for a faithful translation of the objective to a scalar reward, the agent has to learn how to control the automaton's nondeterministic choices; that is, the agent has to learn when the SLDBA should cross from the initial component to the accepting component to produce a successful run of a behavior that satisfies the given objective.

Any GFM automaton with a Büchi acceptance condition can be used instead of an SLDBA in the approach of [12]. While in many cases SLDBAs work well, GFM automata that are not limit-deterministic may provide a significant advantage.

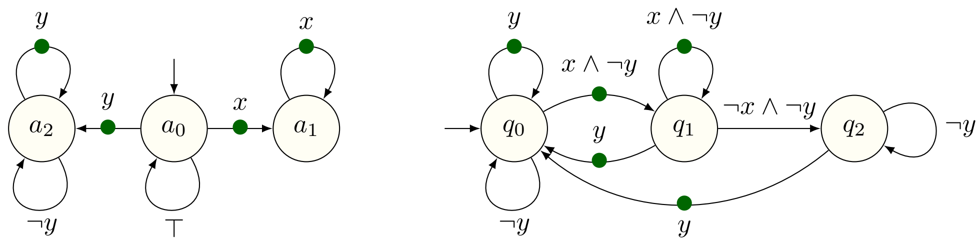


Figure 4 Two GFM automata for $(\mathsf{F}\,\mathsf{G}\,x) \vee (\mathsf{G}\,\mathsf{F}\,y)$. SLDBA (left), and forgiving (right).

Early during training, the agent relies on uniform random choices to discover policies that lead to successful episodes. This includes randomly resolving the automaton nondeterminism. If random choices are unlikely to produce successful runs of the automaton in case of behaviors that should be accepted, learning is hampered because good behaviors are not rewarded. Therefore, GFM automata that are more likely to accept under random choices will result in the agent learning more quickly.

We now discuss three properties of GFM automata that affect the agent's ability to quickly discover correct policies.

Low branching degree. A low branching degree, as mentioned in the introduction, presents the agent with fewer alternatives to pick from. With fewer alternatives to try, the expected number of trials before the agent finds a good combination of choices is lower.

Consider an MDP and an automaton that require that a specific sequence of k nondeterministic choices be made in order for the automaton to accept. If at each choice there are b equiprobable options, the correct sequence is obtained with probability b^{-k}. This is a simplified description of what happens in the `milk` example, described in Appendix B.2. In this example, the agent learns in fewer episodes with a slim automaton than with other GFM automata with higher branching degrees.

Cautiousness. An automaton that enables fewer nondeterministic choices for the same finite input word gives the agent fewer chances to make the wrong choice. The slim automata construction based on breakpoints has the interesting property of "collecting hints of acceptance" before a nondeterministic choice is enabled because S' has to be nonempty for a $\delta_{2,1}$ transition to be present and that requires going through at least one accepting transition.

Consider a model with the objective $\mathsf{F}\,\mathsf{G}\,p$, in which p changes value at each transition many times before a winning end component may be reached. An SLDBA for $\mathsf{F}\,\mathsf{G}\,p$ that may take a nondeterministic transition to the accepting part whenever it reads p is very likely to jump prematurely and reject. However, an automaton produced by the breakpoint construction must see p twice in a row before the jump is enabled, preventing the agent from making mistakes in its control. (See example `oddChocolates` in Appendix B.2.)

Forgiveness. Mistakes made in resolving nondeterminism are often irrecoverable. This is often true of SLDBAs meant for model checking, in which jumps are made to select a subformula to be eventually satisfied. However, general GFM automata, thanks also to their less constrained structure, may be constructed to "forgive mistakes" by giving more chances of picking a successful run.

Figure 4 compares a typical SLDBA to an automaton that is not limit-deterministic and is not produced by the breakpoint construction, but is proved GFM by AEC simulation. This latter automaton has a nondeterministic choice in state q_0 on letter $x \wedge \neg y$ that can be made an unbounded number of times. The agent may choose q_1 repeatedly even if eventually $\mathsf{F}\,\mathsf{G}\,x$ is false and $\mathsf{G}\,\mathsf{F}\,y$ is true. With the SLDBA, on the other hand, there is no room for error. Appendix B.2 presents a model where the added flexibility improves learning.

The introduction of slim automata and AEC simulation extends GFM automata well beyond limit-deterministic automata. This enables the choice of automata tailored to the task at hand, leading to more efficient probabilistic model checking and reinforcement learning.

References

1 T. Babiak, M. Křetínský, V. Rehák, and J. Strejcek. LTL to Büchi automata translation: Fast and more deterministic. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 95–109, 2012.
2 Ch. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
3 C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *Foundations of Computer Science*, pages 338–345. IEEE, 1988.
4 C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, July 1995.
5 L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1998.
6 D. L. Dill, A. J. Hu, and H. Wong-Toi. Checking for language inclusion using simulation relations. In *Computer Aided Verification*, pages 255–265, July 1991. LNCS 575.
7 A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. Spot 2.0 - A framework for LTL and ω-automata manipulation. In *Automated Technology for Verification and Analysis*, pages 122–129, 2016.
8 K. Etessami, T. Wilke, and R. A. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. *SIAM J. Comput.*, 34(5):1159–1175, 2005.
9 K. Etessami, T. Wilke, and R. A. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. *SIAM Journal of Computing*, 34(5):1159–1175, 2005.
10 S. Gurumurthy, R. Bloem, and F. Somenzi. Fair simulation minimization. In *Computer Aided Verification (CAV'02)*, pages 610–623, July 2002. LNCS 2404.
11 E. M. Hahn, G. Li, S. Schewe, A. Turrini, and L. Zhang. Lazy probabilistic model checking without determinisation. In *Concurrency Theory*, pages 354–367, 2015.
12 E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak. Omega-regular objectives in model-free reinforcement learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 395–412, 2019. LNCS 11427.
13 T. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *Concurrency Theory*, pages 273–287, 1997. LNCS 1243.
14 T. A. Henzinger and N. Piterman. Solving games without determinization. In *Computer Science Logic*, pages 394–409, September 2006. LNCS 4207.
15 D. Kini and M. Viswanathan. Optimal translation of LTL to limit deterministic automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 113–129, 2017.
16 J. Klein, D. Müller, Ch. Baier, and S. Klüppelholz. Are good-for-games automata good for probabilistic model checking? In *Language and Automata Theory and Applications*, pages 453–465, 2014.
17 J. Křetínský, T. Meggendorfer, S. Sickert, and Ch. Ziegler. Rabinizer 4: from LTL to your favourite deterministic automaton. In *Computer Aided Verification*, pages 567–577. Springer, 2018.
18 J. Křetínský, T. Meggendorfer, and S. Sickert. Owl: A library for ω-words, automata, and LTL. In *Automated Technology for Verification and Analysis*, pages 543–550, 2018.
19 R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.
20 R. Milner. An algebraic definition of simulation between programs. *Int. Joint Conf. on Artificial Intelligence*, pages 481–489, 1971.
21 S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.
22 N. Piterman. From deterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3):1–21, 2007.
23 M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York, NY, USA, 1994.
24 S. Safra. *Complexity of Automata on Infinite Objects*. PhD thesis, The Weizmann Institute of Science, March 1989.
25 S. Schewe. Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, pages 400–411, 2010.
26 S. Schewe and T. Varghese. Tight bounds for the determinisation and complementation of generalised Büchi automata. In *Automated Technology for Verification and Analysis*, pages 42–56, 2012.

27 S. Schewe and T. Varghese. Determinising parity automata. In *Mathematical Foundations of Computer Science*, pages 486–498, 2014.

28 S. Sickert, J. Esparza, S. Jaax, and J. Křetínský. Limit-deterministic Büchi automata for linear temporal logic. In *Computer Aided Verification*, pages 312–332, 2016. LNCS 9780.

29 S. Sickert and J. Křetínský. MoChiBA: Probabilistic LTL model checking using limit-deterministic Büchi automata. In *Automated Technology for Verification and Analysis*, pages 130–137, 2016.

30 S. Sohail, F. Somenzi, and K. Ravi. A hybrid algorithm for LTL games. In *Verification, Model Checking and Abstract Interpretation*, pages 309–323, San Francisco, CA, January 2008. LNCS 4905.

31 F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Computer Aided Verification*, pages 248–263, July 2000. LNCS 1855.

32 M.-H. Tsai, S. Fogarty, M. Y. Vardi, and Y.-K. Tsay. State of Büchi complementation. *Logical Mehods in Computer Science*, 10(4), 2014.

33 M.-H. Tsai, Y.-K. Tsay, and Y.-S. Hwang. GOAL for games, omega-automata, and logics. In *Computer Aided Verification*, pages 883–889, 2013.

34 M. Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Foundations of Computer Science*, pages 327–338, 1985.

A Omitted Proofs

A.1 Proof of Theorem 6

The most trivial observation is that $\mathcal{S}$ is slim: its set of transitions is the union of two sets of deterministic transitions. We continue by showing that $\mathcal{S}$ simulates $\mathcal{A}$.

In the simulation game, the duplicator can follow the following strategy.

Initial Phase: Every move of the spoiler—with some letter σ—from the transitions δ_S—the subset part of $\mathcal{A}$—is followed by a move from δ_B with the same letter σ. When the duplicator follows this strategy, and when, after a pair of moves, the pebble of the spoiler is on state $S \subseteq Q$, then the pebble of the duplicator is on some state (S, S'). Note that the spoiler loses if she stays forever in this phase.

Transition Phase: The one spoiler move—with some letter σ—from the transitions Δ_{SB}—the transition to the breakpoint part of $\mathcal{A}$—is followed by a move from δ_B with the same letter σ. When the duplicator follows this strategy, and when, after the pair of moves, the pebble of the spoiler is on state $(S, \emptyset)$, then the pebble of the duplicator is on some state (T, T') with $S \subseteq T$.

Final Phase: Every move of the spoiler—with some letter σ—from the transitions δ_B—the breakpoint part of $\mathcal{A}$—is followed by a move from δ_B with the same letter σ, unless (1) the selected spoiler move is from γ_B—and consequently leads to a state of the form $(S, \emptyset)$, i.e. a state where the second set is empty—and (2) since the last time the spoiler has passed by a state where the second set is empty, the duplicator has not passed an accepting transition. When these two conditions are satisfied, then the duplicator uses a transition from $\delta_{2,1}$.

One can now see that, after each pair of moves, the first element of the state (S_1, S_2) the spoiler's pebble is on is contained in the first element of the state (T_1, T_2) the duplicator's pebble is on. The trivial case is to see that this always holds when the duplicator uses a transition from δ_B.

To see that this also holds when the duplicator uses a transition from $\delta_{2,1}$, we start an inductive sub-argument from the previous point in time where the spoiler was in state (S_1, S_2) with $S_2 = \emptyset$. Clearly $S_2 = \emptyset$ entails $S_2 \subseteq T_2$. Now, while the spoiler and the duplicator use a non-accepting transition from δ_B (for some letter σ), the following holds: when the spoiler moves to (S_1', S_2') and the duplicator to (T_1', T_2'), then $S_2' = \delta_S(S_2, \sigma) \cup \gamma_S(S_1, \sigma) \subseteq \delta_S(T_2, \sigma) \cup \gamma_S(T_1, \sigma) = T_2'$. When the spoiler uses a transition from γ_B and the duplicator the transition from $\delta_{2,1}$, the following holds: when the spoiler moves to $(S_1', \emptyset)$ and the duplicator to $(T_1', \emptyset)$, then $S_1' = \delta_S(S_2, \sigma) \cup \gamma_S(S_1, \sigma) \subseteq \delta_S(T_2, \sigma) \cup \gamma_S(T_1, \sigma) = T_1'$.

This allows the duplicator to react on the moves of the spoiler in the way described. It also makes sure that the spoiler cannot produce two consecutive accepting transitions without the duplicator also producing an accepting transition, such that the duplicator accepts when the spoiler accepts. The duplicator therefore wins the simulation game.

Using Lemma 3, it now suffices to show that the language of $\mathcal{S}$ is included in the language of $\mathcal{B}$. To show this, we simply argue that an accepting run $\rho = (Q_0, Q_0'), (Q_1, Q_1'), (Q_2, Q_2'), (Q_3, Q_3'), \ldots$ of $\mathcal{S}$ on an input word $\alpha = \sigma_0, \sigma_1, \sigma_2, \ldots$ can be interpreted as a forest of finitely many finitely branching trees of overall infinite size, where all infinite branches are accepting runs of $\mathcal{B}$. Kőnig's Lemma then provides the existence of an accepting run of $\mathcal{B}$.

This forest is the usual one. The nodes are labeled by states of $\mathcal{B}$, and the roots (level 0) are the initial states of $\mathcal{B}$. Let $I = \Big\{ i \in \mathbb{N} \mid \big((Q_{i-1}, Q_{i-1}'), \sigma_{i-1}, (Q_i, Q_i')\big) \in \Gamma := \mathsf{ndet}(\gamma_B) \cup \mathsf{ndet}(\delta_{2,1}) \Big\}$ be the set of positions after accepting transitions in ρ. We define the predecessor function $\mathsf{pred}\colon \mathbb{N} \to I \cup \{0\}$ with $\mathsf{pred}\colon i \mapsto \max\big\{ j \in I \cup \{0\} \mid j < i \big\}$.

We call a node with label q_l on level l an end-point if one of the following applies: (1) $q_l \notin Q_l$ or (2) $l \in I$ and for all j such that $\mathsf{pred}(l) \leq j < l$, where q_j is the label of the ancestor of this node on level j, we have $(q_j, \sigma_j, q_{j+1}) \notin \Gamma$.

(1) may only happen after a transition from $\delta_{2,1}$ has been taken, and the q_l is not among the states that is traced henceforth. (2) identifies parts of the run tree that do not contain an accepting transition.

A node labeled with q_l on level l that is not an endpoint has $\left|\delta_S(q_l, \sigma_l)\right|$ children, labeled with the different elements of $\delta_S(q_l, \sigma_l)$. It is now easy to show by induction over i that the following hold.

1. For all $q \in Q_i$, there is a node on level i labeled with q.
2. For $i \notin I$ and all $q \in Q'_i$, there is a node labeled q on level i, a j with $\mathsf{pred}(i) \leq j < i$, and ancestors on level j and $j+1$ labeled q_j and q_{j+1}, such that $(q_j, \sigma_j, q_{j+1}) \in \Gamma$. (The 'ancestor' on level $j+1$ might be the state itself.)
   For $i \in I$ and all $q \in Q'_i$, there is a node labeled q on level i, which is not an end point.

Consequently, the forest is infinite, finitely branching, and finitely rooted, and thus contains an infinite path. By construction, this path is an accepting run of $\mathcal{B}$.

A.2 Proof of Theorem 8

For the proof, we use an arbitrary (but fixed) MDP $\mathcal{M}$, and an arbitrary (but fixed) pure optimal positional strategy μ for $\mathcal{M} \times \mathcal{B}$, resulting in the Markov chain $(\mathcal{M} \times \mathcal{B})_\mu$. We assume w.l.o.g. that the accepting LSCCs in $(\mathcal{M} \times \mathcal{B})_\mu$ are identified, e.g., by a bit.

Let τ be a winning strategy of the duplicator against the spoiler in an ℓ-GAEC simulation game. Abusing notation, we let $\tau \circ \mu$ denote the finite-memory strategy[3] obtained from μ and τ for $\mathcal{M} \times \mathcal{A}$, where τ is acting only on the automata part of $(\mathcal{M} \times \mathcal{B})$, and where the spoiler makes the move to the end-component when she is in some LSCC B of $(\mathcal{M} \times \mathcal{B})_\mu$ and gives the full list of sequences of transitions of length ℓ that occur in B.

Proof. As $\mathcal{B}$ is good for MDPs, we only have to show that the chance of winning in $(\mathcal{M} \times \mathcal{A})_{\tau \circ \mu}$ is at least the chance of winning in $(\mathcal{M} \times \mathcal{B})_\mu$. The chance of winning in $(\mathcal{M} \times \mathcal{B})_\mu$ is the chance of reaching an accepting LSCC in $(\mathcal{M} \times \mathcal{B})_\mu$. It is also the chance of reaching an accepting LSCC in $(\mathcal{M} \times \mathcal{B})_\mu$ *and*, after reaching a LSCC L, to see exactly those sequences of transitions of length ℓ that occur in L, and to see all of them infinitely often.

By construction, $\tau \circ \mu$ will translate those runs into accepting runs of $(\mathcal{M} \times \mathcal{A})_{\tau \circ \mu}$, such that the chances of an accepting run of $(\mathcal{M} \times \mathcal{A})_{\tau \circ \mu}$ is at least the chance of an accepting run of $(\mathcal{M} \times \mathcal{B})_\mu$. As μ is optimal, the chance of winning in $\mathcal{M} \times \mathcal{A}$ is at least the chance of winning in $\mathcal{M} \times \mathcal{B}$. As $\mathcal{B}$ is good for MDPs, this is the chance of $\mathcal{M}$ producing a run accepted by $\mathcal{B}$ (and thus $\mathcal{A}$) when controlled optimally, which is an upper bound on the chance of winning in $\mathcal{M} \times \mathcal{A}$. ◀

A.3 Proof of Theorem 10

In both cases, the rules to translate a winning strategy of the spoiler for the 1-GAEC simulation game is straight forward: The spoiler can essentially follow her winning strategy from the 1-GAEC simulation game, with the extra rule that she will make her AEC claim to the duplicator on the first accepting transition on or after her AEC claim in the 1-GAEC claim. In case the duplicator is allowed to update the transition, this information is ignored by the spoiler—she plays in accordance with her winning strategy from the 1-GAEC simulation game.

Naturally, the resulting play will comply with her 1-GAEC claim, and will thus also be winning for the—weaker—AEC claim made to the duplicator.

[3] The strategy τ consists of one sub-strategy to be used before the AEC claim is made and one sub-strategy for each possible ℓ-GAEC claim. The memory of $\tau \circ \mu$ keeps track of the position in $(\mathcal{M} \times \mathcal{B})_\mu$. When an accepting LSCC is detected (via the marker bit) then analysis of $(\mathcal{M} \times \mathcal{B})_\mu$ reveals the only possible ℓ-GAEC claim. This claim is used to select the right entry from τ.

B Results

B.1 Simulation Results with Random LTL Formulas

Table 1 Simulation results. Column “LTL” contains the formula the automata are generated from. Column “sim” contains the type of simulation required to show that the original automaton is suitable for MDPs. There, “det” means that the automaton is deterministic (no simulation required), “sim0” means that standard simulation sufficed, “sim1” means that the simulation where the spoiler is forced to choose a transition to be repeated infinitely often suffices, “sim2” means that in addition the witness can change the transition which has to occur infinitely often, and “nosim” means that we were not able to prove a simulation relation. “ltl2tgba” contains the number of states from SPOT’s ltl2tgba, “owl” the ones from Owl’s ltl2ldba, and “owl-nd” the ones from ltl2ldba with nondeterminism in the initial part.

ltl	sim	ltl2tgba	owl	owl-nd
$(\mathsf{XF}(ap_3 \rightarrow (\neg ap_0 \vee \neg \mathsf{G} \neg ap_2)) \wedge \neg((ap_1 \rightarrow ap_3)\mathsf{RG}ap_1)) \vee ((\mathsf{F}ap_2 \mathsf{W} ap_1) \wedge (\mathsf{G}ap_3 \mathsf{M} ap_3))$	det	7	14	21
$\neg\mathsf{G}(ap_1 \mathsf{W} ap_2) \rightarrow ((ap_3 \wedge (ap_1 \rightarrow ap_3)) \leftrightarrow \neg \mathsf{F} ap_2)$	sim0	5	8	6
$((ap_2 \rightarrow \mathsf{XX}ap_3) \rightarrow \neg\mathsf{X}((ap_1 \mathsf{xor} \mathsf{F}ap_2) \leftrightarrow ((ap_0 \mathsf{RG} ap_3)\mathsf{M}\, ap_3))) \rightarrow ((ap_0 \wedge (ap_3 \mathsf{R}((ap_0 \mathsf{xor} \mathsf{X} ap_1) \vee (\mathsf{X}\neg(ap_1 \leftrightarrow ap_3) \rightarrow \mathsf{F}ap_3)))) \mathsf{MX} ap_2)$	sim1	24	84	75
$(\neg\mathsf{XF}\neg ap_2 \mathsf{xor} \neg\mathsf{G}(ap_3 \rightarrow \mathsf{F}ap_0))\mathsf{WF}(((\mathit{ff}\, \mathsf{RX}ap_1)\mathsf{M}(ap_2 \leftrightarrow \mathsf{G}\, (ap_2 \leftrightarrow (ap_1\ \mathsf{U}\ \neg ap_1))))\ \mathsf{U}\ \neg(ap_0 \mathsf{W} ap_1))$	sim2	17	35	28
$\neg(((ap_3 \rightarrow (ap_2 \leftrightarrow ap_3))\mathsf{M}ap_1)\mathsf{R}ap_3)\mathsf{W}\neg\mathsf{FG}ap_1$	nosim	5	9	9

B.2 Reinforcement Learning Case Studies

We collect here the models and properties discussed in Section 4.2. The `milk` model, shown below as PRISM[4] code, shows the effects of low branching degree. The property of interest is $\bigwedge_{0 \le i \le 4} \mathsf{F}\,\mathsf{G}\, p_i$. The slim automaton has 275 states, while the SLDBA automaton has 7 states. Since the p_i’s are not mutually exclusive, the slim automaton needs a large number of states to limit the branching degree. Yet, for a fixed set of hyperparameter values, the agent learns the optimal policy with the slim automaton and fails to learn it with the SLDBA.

```
mdp
const int M = 12;

module m
  b : [0..M] init M;
  [a] true → (b' = b > 0 ? b−1 : b);
endmodule

label "p0" = b > 0;
label "p1" = b > 1;
label "p2" = b > 2;
label "p3" = b > 3;
label "p4" = b = 0;
```

[4] M. Kwiatkowska, G. Norman, and D. Parker, PRISM 4.0: Verification of Probabilistic Real-time Systems, Computer Aided Verification, pp. 585–591, 2011. LNCS 6806.

The `oddChocolates` model shows the benefits of cautiousness in resolving nondeterminism. The property is $\mathsf{F}\,\mathsf{G}\,\mathsf{odd}$. The slim automaton has 4 states and is limit-deterministic, while the usual SLDBA, optimized for size, has 3 states. The extra state produced by the breakpoint construction allows the slim automaton to jump to the accepting part only after seeing two consecutive "odd" states. As a result, for a fixed set of hyperparameter values, and the tool MUNGOJERRIE [12], the agent reliably learns an optimal policy, unlike the case of the smaller SLDBA automaton.

```
mdp
const int N = 5;
const int M = 12;

module pluck
  b0 : [0..M] init M;
  b1 : [0..M] init M;
  b2 : [0..M] init M;
  b3 : [0..M] init M;
  b4 : [0..M] init M;
  [a0] true → (b0' = b0 > 0 ? b0−1 : b0);
  [a1] true → (b1' = b1 > 0 ? b1−1 : b1);
  [a2] true → (b2' = b2 > 0 ? b2−1 : b2);
  [a3] true → (b3' = b3 > 0 ? b3−1 : b3);
  [a4] true → (b4' = b4 > 0 ? b4−1 : b4);
endmodule

label "odd" = mod(b0+b1+b2+b3+b4,2) = 1;
```

The `forgiveness` model below illustrates the effects of forgiveness. Its main feature is to produce, while in its transient states, deceiving hints for the learner. Two automata for $(\mathsf{F}\,\mathsf{G}\,x) \vee (\mathsf{G}\,\mathsf{F}\,y)$ are shown in Figure 4. The slim automaton produced by the breakpoint construction is not limit-deterministic. It accepts behaviors that satisfy $\mathsf{G}\,\mathsf{F}\,y$ before (irreversibly) jumping and behaviors that satisfy $\mathsf{F}\,\mathsf{G}\,x$ after jumping. Therefore the agent can recover from mistakenly believing that an MDP run will satisfy $\mathsf{G}\,\mathsf{F}\,y$, while it ends up satisfying $\mathsf{F}\,\mathsf{G}\,x$, but not vice versa. Accordingly, the results with the slim automaton are in between those of the two automata of Figure 4.

```
mdp
const double p = 1/2;
const double q = 0.1;
const int N = 5;

module m
  s : [0..7] init 0;
  d : [0..N] init N;
  [a] s=0 → p : (s'=1) + (1−p) : (s'=2);
  [b] s=0 → p : (s'=2) + (1−p) : (s'=3);
  [c] (s=1 ∨ s=2) ∧ d > 0 → q : (d'=d−1) + (1−q) : true;
  [c] (s=1 ∨ s=2) ∧ d = 0 → (s'=s+3);
  [c] s=3 → true;
  [a] s=4 ∨ s=5 → q : (s'=s+2) + (1−q) : true;
  [b] s=4 ∨ s=5 → true;
  [c] s=6 ∨ s=7 → (s'=s−2);
endmodule

label "x" = s=1 ∨ s=5;
label "y" = s=2 ∨ s=6;
```

B.3 Discussion

We have defined the class of automata that are *good for MDPs*—nondeterministic automata that can be used for the analysis of MDPs—and shown it to be closed under different simulation relations. This has multiple implications that look very promising for model checking and reinforcement learning for MDPs. Closure under classic simulation already opens a rich toolbox of statespace reduction techniques that always come in handy to push the boundary of analysis techniques, while the more powerful (and more expensive) AEC simulation has promise to identify situations, where source automata are coincidentally good for MDPs. Using standard and (very basic) AEC simulation, we only failed to establish the good for MDPs property for the NBAs created for 108 of 685 random LTL examples that are not expressible as DBAs, 72 of them due to timeout.

The wider class of GFM automata also shows promise: the slim automata we have defined to tame the branching degree while retaining the desirable Büchi condition for reinforcement learning are already attractive. They are comparable in their simplicity to the SLDBAs from the 80s [34, 3]. To our surprise they fared very well and were able to compete even when evaluated against optimized SLDBAs, outperforming them significantly on our case studies.

As outlined in Section 4.2, the experimental results detailed in this appendix highlight three properties of automata that make them particularly well-suited for learning:

1. a low branching degree;
2. cautiousness; and
3. forgiveness.

A low branching degree is the easiest among them to quantify, and the *slim automata* tick this box perfectly. We have also argued that they are contributing towards cautiousness. For forgiveness, on the other hand, we had to hand-craft automata that satisfy this property.

From a practical point of view, much of the power of this new approach is in harnessing the power of simulation for learning, and forgiveness is closely related to simulation. For the automata from Figure 4, the state q_0 from the forgiving automaton on the right simulates all three states of the SLDBA on the left, q_1 simulates a_1 and a_2, and q_2 simulates a_2, but no state of the the SLDBA simulates any state of the forgiving automaton.

The natural follow-up research will be to test and tap the full potential of simulation based statespace reduction instead of the limited version—which preserves limit determinism—that we have currently implemented. Besides using this to get the statespace small—which is particularly useful for model checking—we will study the use simulation to construct forgiving automata, which is particularly promising for reinforcement learning.

C Proving Simulation

C.1 Classic Simulation

In the following, we discuss how we can prove that a given Büchi automaton $\mathcal{D} = \langle \Sigma, Q_\mathcal{D}, Q_{\mathcal{D},0}, \Delta_\mathcal{D}, \Gamma_\mathcal{D} \rangle$ (duplicator) simulates a Büchi automaton $\mathcal{S} = \langle \Sigma, Q_\mathcal{S}, Q_{\mathcal{S},0}, \Delta_\mathcal{S}, \Gamma_\mathcal{S} \rangle$ (spoiler). The basic idea is to construct a parity game $\mathcal{G}$ such that the even player wins if and only if the simulation holds. The construction of the game is fairly standard and closely follows the outline from Subsection 3.1.

In our notion, we define parity games as follows.

▶ **Definition 11.** *A* transition-labelled maximum parity game *is a tuple*

$$\mathcal{P} = \langle Q_0, Q_1, \Delta, c \rangle$$

where for $Q = Q_0 \cup Q_1$ *we have*

- $Q_0 \cap Q_1 = \emptyset$,
- $\Delta \subseteq Q \times Q$,
- $c\colon Q \times Q \rightharpoonup \mathbb{N}$ *where* $Dom(c) = \Delta$.

An infinite play *is an infinite sequence* $p = q_0 q_1 \ldots \in Q^\omega$ *such for for all* $i \geq 0$ *it is* $(q_i, q_{i+1}) \in \Delta$. *An infinite play* p *is* winning *if the highest colour which appears infinitely often is even, that is* $\max\{c' \mid \forall i \geq 0 \exists j \geq i.\ c' = c(q_j, q_{j+1})\} \mod 2 = 0$. *By* $\mathrm{infplays}(\mathcal{P})$ *we denote the set of all infinite plays of* $\mathcal{P}$. *A* finite play *is a finite sequence* $p = q_0 q_1 \ldots q_n \in Q^*$ *such for for all* $i, 0 \leq i \leq n-1$ *it is* $(q_i, q_{i+1}) \in \Delta$. *By* $\mathrm{finplays}_i(\mathcal{P})$ *we denote the set of all finite plays* p *of* $\mathcal{P}$ *such that* $q_n \in Q_i$ *for* $i \in \{0, 1\}$. *A* strategy *for player* i *is a function* $f_i\colon \mathrm{finplays}_i(\mathcal{P}) \to Q$ *such that for* $p = q_0 q_1 \ldots q_n$ *we have* $(q_n, f(p)) \in \Delta$. *Given a pair of a player* 0 *and player* 1 *strategies* $\langle f_0, f_1 \rangle$, *the* induced play *starting from state* q *is* $p = q_0 q_1 q_2 \ldots$ *where* $q_0 = q$ *and for all* $k \geq 0$ *we have* $q_{k+1} = f_i(q_k)$ *where* $q_k \in \Delta_i$. *A player* 0 *strategy* f_0 *is* winning *in a state* q *if for every player* 1 *strategy* f_1 *the induced play is winning.*

Note that here colours are attached to transitions. This change is only for technical convenience, because colours naturally occur on the edges rather than states of the games we construct. Models of this type can either be transformed to state-labelled parity games or solved by slightly adapted versions of existing algorithms.

The parity game we construct is then as follows:

▶ **Definition 12.** *Consider two Büchi automata* $\mathcal{S} = \langle \Sigma, Q_\mathcal{S}, Q_{\mathcal{S},0}, \Delta_\mathcal{S}, \Gamma_\mathcal{S} \rangle$ *and* $\mathcal{D} = \langle \Sigma, Q_\mathcal{D}, Q_{\mathcal{D},0}, \Delta_\mathcal{D}, \Gamma_\mathcal{D} \rangle$. *The* classical simulation parity game $\mathcal{P}_{\mathcal{S},\mathcal{D},\mathrm{sim0}}$ *is defined as*

$$\mathcal{P}_{\mathcal{S},\mathcal{D},\mathrm{sim0}} = \langle Q_0, Q_1, \Delta, c \rangle$$

where

- $Q_0 = Q_\mathcal{S} \times Q_\mathcal{D} \times \Sigma$,
- $Q_1 = Q_\mathcal{S} \times Q_\mathcal{D}$,
- $\Delta = \Delta_1 \cup \Delta_0$ *where*
- $\Delta_0 = \{((q_\mathcal{S}, q_\mathcal{D}, a), (q_\mathcal{S}, q'_\mathcal{D})) \mid (q_\mathcal{D}, a, q'_\mathcal{D}) \in \Delta_\mathcal{D}\}$
- $c((q_\mathcal{S}, q_\mathcal{D}, a), (q_\mathcal{S}, q'_\mathcal{D})) = \begin{cases} 2 & (q_\mathcal{D}, a, q'_\mathcal{D}) \in \Gamma_\mathcal{D} \\ 0 & \textit{else} \end{cases}$
- $\Delta_1 = \{((q_\mathcal{S}, q_\mathcal{D}), (q'_\mathcal{S}, q_\mathcal{D}, a)) \mid (q_\mathcal{S}, a, q'_\mathcal{S}) \in \Delta_\mathcal{S}\}$
- $c((q_\mathcal{S}, q_\mathcal{D}), (q'_\mathcal{S}, q_\mathcal{D}, a)) = \begin{cases} 1 & (q_\mathcal{S}, a, q'_\mathcal{S}) \in \Gamma_\mathcal{S} \\ 0 & \textit{else} \end{cases}$

The states $Q_1 = Q_\mathcal{S} \times Q_\mathcal{D}$ are the states controlled by the spoiler. A state $(q_\mathcal{S}, q_\mathcal{D})$ represents the situation where the spoiler is in state $q_\mathcal{S}$ and the duplicator is in state $q_\mathcal{D}$. States $Q_0 = Q_\mathcal{S} \times Q_\mathcal{D} \times \Sigma$ are controlled by the duplicator. A state $(q_\mathcal{S}, q_\mathcal{D}, a)$ represents the situation where the spoiler has chosen a transition with label a and has moved to state $q_\mathcal{S}$ then; the component a has to be included in the state description, because the duplicator has to react with a transition with the same label as the one chosen by the spoiler. The transitions of Δ_1 are the ones of the spoiler. The spoiler can choose any input a from its current state $q_\mathcal{S}$ and then move to its successor state $q'_\mathcal{S}$ if $(q_\mathcal{S}, a, q'_\mathcal{S})$ is a valid transition in the spoiler automaton. If this transition is an accepting transition (in the spoiler automaton), then it will have colour 1, otherwise 0. The transitions of Δ_0 are the ones of the duplicator. In a state $(q_\mathcal{S}, q_\mathcal{D}, a)$, the duplicator has to choose a transition (in the Büchi automaton) labelled with a, because this was the choice of input of the spoiler. If this transition is accepting, the transition in the game will have label 2, otherwise it will have label 0.

Now, a play of the constructed parity game exactly describes a simulation game of Subsection 3.1. Also, the play is winning if and only if the simulation game is winning: If the duplicator wins the game, then either a) neither the spoiler nor the duplicator have infinitely many accepting transitions or b) the duplicator has infinitely many accepting transitions. In case a), the resulting colour of the play will be 0. In case b), the resulting colour will be 2. In both cases, it is even and the even player wins. Because of this, we have the following:

▶ **Lemma 13.** *Consider Büchi automata $\mathcal{S} = \langle \Sigma, Q_{\mathcal{S}}, Q_{\mathcal{S},0}, \Delta_{\mathcal{S}}, \Gamma_{\mathcal{S}} \rangle$ and $\mathcal{D} = \langle \Sigma, Q_{\mathcal{D}}, Q_{\mathcal{D},0}, \Delta_{\mathcal{D}}, \Gamma_{\mathcal{D}} \rangle$ and the classical simulation parity game $\mathcal{P}_{\mathcal{S},\mathcal{D}}$. Assume that for all $q_{\mathcal{S}} \in Q_{\mathcal{S},0}$ we have $q_{\mathcal{D}} \in Q_{\mathcal{D},0}$ such that $(q_{\mathcal{S}}, q_{\mathcal{D}})$ is a winning state in the parity game. Then $\mathcal{D}$ simulates $\mathcal{S}$. Otherwise, $\mathcal{D}$ does not simulate $\mathcal{S}$.*

To use the above lemma to show that simulation holds, the usual algorithms for solving parity games can be used (with a slight adaption to take into account the fact that the edges are labelled with a parity rather than the states). We have implemented a variation of the McNaughton algorithm [19]. Note that the parity games we construct have no more than three colours, for which a specialised algorithms exists [9]. However, because in our experience it always took much more time to construct the parity game than to solve this, we did not apply this specialised algorithm so far.

C.2 AEC Simulation

Classic simulation is not required to preserve good for MDP-ness, and the above parity game might fail to demonstrate a good for MDP property. In Subsection 3.3, we have discussed of how the simulation can be refined such that more automata can be proven to be sufficiently similar for our purpose (AEC simulation). In the next definition, we force the spoiler to choose a transition which is to be repeated infinitely often.

▶ **Definition 14.** *Consider two Büchi automata $\mathcal{S} = \langle \Sigma, Q_{\mathcal{S}}, Q_{\mathcal{S},0}, \Delta_{\mathcal{S}}, \Gamma_{\mathcal{S}} \rangle$ and $\mathcal{D} = \langle \Sigma, Q_{\mathcal{D}}, Q_{\mathcal{D},0}, \Delta_{\mathcal{D}}, \Gamma_{\mathcal{D}} \rangle$. The* accepting transition simulation parity game *$\mathcal{P}_{\mathcal{S},\mathcal{D},\mathrm{sim1}}$ is defined as*

$$\mathcal{P}_{\mathcal{S},\mathcal{D},\mathrm{sim1}} = \langle Q_0, Q_1, \Delta, c \rangle$$

where

- $Q_0 = Q_{\mathcal{S}} \times Q_{\mathcal{D}} \times \Sigma \cup Q_{\mathcal{S}} \times Q_{\mathcal{D}} \times \Sigma \times (Q_{\mathcal{S}} \times \Sigma \times Q_{\mathcal{S}})$,
- $Q_1 = Q_{\mathcal{S}} \times Q_{\mathcal{D}} \cup Q_{\mathcal{S}} \times Q_{\mathcal{D}} \times (Q_{\mathcal{S}} \times \Sigma \times Q_{\mathcal{S}})$,
- $\Delta = \Delta_{0,i} \cup \Delta_{1,i} \cup \Delta_{1,t} \cup \Delta_{0,f} \cup \Delta_{1,f}$ *where*
- $\Delta_{0,i} = \{((q_{\mathcal{S}}, q_{\mathcal{D}}, a), (q_{\mathcal{S}}, q'_{\mathcal{D}})) \mid (q_{\mathcal{D}}, a, q'_{\mathcal{D}}) \in \Delta_{\mathcal{D}}\}$
- $c((q_{\mathcal{S}}, q_{\mathcal{D}}, a), (q_{\mathcal{S}}, q'_{\mathcal{D}})) = 0$
- $\Delta_{1,i} = \{((q_{\mathcal{S}}, q_{\mathcal{D}}), (q'_{\mathcal{S}}, q_{\mathcal{D}}, a)) \mid (q_{\mathcal{S}}, a, q'_{\mathcal{S}}) \in \Delta_{\mathcal{S}}\}$,
- $c((q_{\mathcal{S}}, q_{\mathcal{D}}), (q'_{\mathcal{S}}, q_{\mathcal{D}}, a)) = 0$,
- $\Delta_t = \{((q_{\mathcal{S}}, q_{\mathcal{D}}), (q'_{\mathcal{S}}, q_{\mathcal{D}}, a, (q_{\mathcal{S}}, a, q'_{\mathcal{S}}))) \mid (q_{\mathcal{S}}, a, q'_{\mathcal{S}}) \in \Gamma_{\mathcal{S}}\}$
- $c((q_{\mathcal{S}}, q_{\mathcal{D}}), (q'_{\mathcal{S}}, q_{\mathcal{D}}, a, (q_{\mathcal{S}}, a, q'_{\mathcal{S}}))) = 0$,
- $\Delta_{0,f} = \{((q_{\mathcal{S}}, q_{\mathcal{D}}, a, e), (q_{\mathcal{S}}, q'_{\mathcal{D}}, e)) \mid (q_{\mathcal{D}}, a, q'_{\mathcal{D}}) \in \Delta_{\mathcal{D}}\}$
- $c((q_{\mathcal{S}}, q_{\mathcal{D}}, a, e), (q_{\mathcal{S}}, q'_{\mathcal{D}}, e)) = \begin{cases} 2 & (q_{\mathcal{D}}, a, q'_{\mathcal{D}}) \in \Gamma_{\mathcal{D}} \\ 0 & \textit{else} \end{cases}$
- $\Delta_{1,f} = \{((q_{\mathcal{S}}, q_{\mathcal{D}}, e), (q'_{\mathcal{S}}, q_{\mathcal{D}}, a, e)) \mid (q_{\mathcal{S}}, a, q'_{\mathcal{S}}) \in \Delta_{\mathcal{S}}\}$
- $c((q_{\mathcal{S}}, q_{\mathcal{D}}, e), (q'_{\mathcal{S}}, q_{\mathcal{D}}, a, e)) = \begin{cases} 1 & (q_{\mathcal{S}}, a, q'_{\mathcal{S}}) = e \\ 0 & \textit{else} \end{cases}$

For the two players, there are now two different types of states. The first type is like the state for the classical simulation parity game. The second type is extended by $(Q_{\mathcal{S}} \times \Sigma \times Q_{\mathcal{S}})$, such that it can store edges of the spoiler. There are now three types of transitions: $\Delta_{0,i}$ and $\Delta_{1,i}$ are similar to the ones of Δ_0 and Δ_0 of the classical simulation parity game. The difference is that the colour is always 0. This implies that the spoiler cannot win by just using this type of transitions. Instead, she has to choose a transition of Δ_t, which corresponds to choosing an edge to occur infinitely often. To reduce the size of the state-space and to ease definition, we restrict this choice such that it can only choose accepting edges, because it cannot win with non-accepting edges anyway. For the same reason, we only allow choosing the edge it just took. Transitions $\Delta_{0,f}$ and $\Delta_{1,f}$ correspond to the situation where the spoiler has already chosen an edge. Transitions of $\Delta_{0,f}$ are labelled with 2 if they correspond to an accepting transition of the duplicator and 0 otherwise. Transitions of $\Delta_{1,f}$ are labelled with 1 if the edge is taken which the spoiler chose to appear infinitely often.

▶ **Lemma 15.** *Consider Büchi automata $\mathcal{S} = \langle \Sigma, Q_\mathcal{S}, Q_{\mathcal{S},0}, \Delta_\mathcal{S}, \Gamma_\mathcal{S}\rangle$ and $\mathcal{D} = \langle \Sigma, Q_\mathcal{D}, Q_{\mathcal{D},0}, \Delta_\mathcal{D}, \Gamma_\mathcal{D}\rangle$ and the accepting simulation parity game $\mathcal{P}_{\mathcal{S},\mathcal{D}}$. Assume that for all $q_\mathcal{S} \in Q_{\mathcal{S},0}$ we have $q_\mathcal{D} \in Q_{\mathcal{D},0}$ such that $(q_\mathcal{S}, q_\mathcal{D})$ is a winning state in the parity game. Then $\mathcal{D}$ accepting transition simulates $\mathcal{S}$. Otherwise, $\mathcal{D}$ does not simulate $\mathcal{S}$.*

Lemma 15 can be used to prove or disprove simulation in the same way as Lemma **??**.

The simulation relation in which the duplicator is allowed to change the edge to be repeated infinitely often can be implemented in the following parity game:

▶ **Definition 16.** *Consider two Büchi automata $\mathcal{S} = \langle \Sigma, Q_\mathcal{S}, Q_{\mathcal{S},0}, \Delta_\mathcal{S}, \Gamma_\mathcal{S}\rangle$ and $\mathcal{D} = \langle \Sigma, Q_\mathcal{D}, Q_{\mathcal{D},0}, \Delta_\mathcal{D}, \Gamma_\mathcal{D}\rangle$. The* accepting transition with update simulation parity game *$\mathcal{P}_{\mathcal{S},\mathcal{D},\mathrm{sim2}}$ is defined as*

$$\mathcal{P}_{\mathcal{S},\mathcal{D},\mathrm{sim2}} = \langle Q_0, Q_1, \Delta, c\rangle$$

where

- $Q_0 = Q_\mathcal{S} \times Q_\mathcal{D} \times \Sigma \cup Q_\mathcal{S} \times Q_\mathcal{D} \times (Q_\mathcal{S} \times \Sigma \times Q_\mathcal{S}) \times (Q_\mathcal{S} \times \Sigma \times Q_\mathcal{S})$,
- $Q_1 = Q_\mathcal{S} \times Q_\mathcal{D} \cup Q_\mathcal{S} \times Q_\mathcal{D} \times (Q_\mathcal{S} \times \Sigma \times Q_\mathcal{S})$,
- $\Delta = \Delta_{0,i} \cup \Delta_{1,i} \cup \Delta_{1,t} \cup \Delta_{0,f} \cup \Delta_e \cup \Delta_{1,f}$ *where*
- $\Delta_{0,i} = \{((q_\mathcal{S}, q_\mathcal{D}, a), (q_\mathcal{S}, q'_\mathcal{D})) \mid (q_\mathcal{D}, a, q'_\mathcal{D}) \in \Delta_\mathcal{D}\}$
- $c((q_\mathcal{S}, q_\mathcal{D}, a), (q_\mathcal{S}, q'_\mathcal{D})) = 0$,
- $\Delta_{1,i} = \{((q_\mathcal{S}, q_\mathcal{D}), (q'_\mathcal{S}, q_\mathcal{D}, a)) \mid (q_\mathcal{S}, a, q'_\mathcal{S}) \in \Delta_\mathcal{S}\}$,
- $c((q_\mathcal{S}, q_\mathcal{D}), (q'_\mathcal{S}, q_\mathcal{D}, a)) = 0$
- $\Delta_t = \{((q_\mathcal{S}, q_\mathcal{D}), (q'_\mathcal{S}, q_\mathcal{D}, (q_\mathcal{S}, a, q'_\mathcal{S}), (q_\mathcal{S}, a, q'_\mathcal{S}))) \mid (q_\mathcal{S}, a, q'_\mathcal{S}) \in \Gamma_\mathcal{S}\}$
- $c((q_\mathcal{S}, q_\mathcal{D}), 0, (q'_\mathcal{S}, q_\mathcal{D}, (q_\mathcal{S}, a, q'_\mathcal{S}), (q_\mathcal{S}, a, q'_\mathcal{S}))) = 0$,
- $\Delta_{0,f} = \{((q_\mathcal{S}, q_\mathcal{D}, (\hat{q_\mathcal{S}}, a, \hat{q_\mathcal{S}}), e), (q_\mathcal{S}, q'_\mathcal{D}, e)) \mid (q_\mathcal{D}, a, q'_\mathcal{D}) \in \Delta_\mathcal{D}\}$,
- $c((q_\mathcal{S}, q_\mathcal{D}, (\hat{q_\mathcal{S}}, a, \hat{q_\mathcal{S}}), e), (q_\mathcal{S}, q'_\mathcal{D}, e)) = \begin{cases} 2 & (q_\mathcal{D}, a, q'_\mathcal{D}) \in \Gamma_\mathcal{D} \\ 0 & else \end{cases}$
- $\Delta_e = \{((q_\mathcal{S}, q_\mathcal{D}, (\hat{q_\mathcal{S}}, a, \hat{q_\mathcal{S}}), e), (q_\mathcal{S}, q'_\mathcal{D}, (\hat{q_\mathcal{S}}, a, \hat{q_\mathcal{S}}))) \mid (q_\mathcal{D}, a, q'_\mathcal{D}) \in \Delta_\mathcal{D}\}$,
- $c((q_\mathcal{S}, q_\mathcal{D}, (\hat{q_\mathcal{S}}, a, \hat{q_\mathcal{S}}), e), (q_\mathcal{S}, q'_\mathcal{D}, (\hat{q_\mathcal{S}}, a, \hat{q_\mathcal{S}}))) = \begin{cases} 2 & (q_\mathcal{D}, a, q'_\mathcal{D}) \in \Gamma_\mathcal{D} \\ 0 & else \end{cases}$
- $\Delta_{1,f} = \{((q_\mathcal{S}, q_\mathcal{D}, e), (q'_\mathcal{S}, q_\mathcal{D}, (q_\mathcal{S}, a, q'_\mathcal{S}), e)) \mid (q_\mathcal{S}, a, q'_\mathcal{S}) \in \Delta_\mathcal{S}\}$,
- $c((q_\mathcal{S}, q_\mathcal{D}, e), (q'_\mathcal{S}, q_\mathcal{D}, (q_\mathcal{S}, a, q'_\mathcal{S}), e)) = \begin{cases} 1 & (q_\mathcal{S}, a, q'_\mathcal{S}) = e \\ 0 & else \end{cases}$

Compared to the accepting transition simulation parity game, we have changed the second type of duplicator states. The spoiler usually chooses an input, and then proceeds to a successor state with this input. Rather than just storing the input, we store the whole edge which the spoiler chose. This allows to change the edge which is expected to be seen infinitely often for the spoiler to win by using the transitions of type Δ_e.

▶ **Lemma 17.** *Consider Büchi automata $\mathcal{S} = \langle \Sigma, Q_\mathcal{S}, Q_{\mathcal{S},0}, \Delta_\mathcal{S}, \Gamma_\mathcal{S}\rangle$ and $\mathcal{D} = \langle \Sigma, Q_\mathcal{D}, Q_{\mathcal{D},0}, \Delta_\mathcal{D}, \Gamma_\mathcal{D}\rangle$ and the accepting simulation with update parity game $\mathcal{P}_{\mathcal{S},\mathcal{D}}$. Assume that for all $q_\mathcal{S} \in Q_{\mathcal{S},0}$ we have $q_\mathcal{D} \in Q_{\mathcal{D},0}$ such that $(q_\mathcal{S}, q_\mathcal{D})$ is a winning state in the parity game. Then $\mathcal{D}$ accepting transition with update simulates $\mathcal{S}$. Otherwise, $\mathcal{D}$ does not simulate $\mathcal{S}$.*

Again, Lemma 17 can be used in the same way as Lemma 15.