Index Appearance Record for Transforming Rabin Automata into Parity Automata

Jan Křetínský(✉), Tobias Meggendorfer, Clara Waldmann, and Maximilian Weininger

Technical University of Munich, Munich, Germany
jan.kretinsky@tum.de

Abstract. Transforming deterministic ω-automata into deterministic parity automata is traditionally done using variants of appearance records. We present a more efficient variant of this approach, tailored to Rabin automata, and several optimizations applicable to all appearance records. We compare the methods experimentally and find out that our method produces smaller automata than previous approaches. Moreover, the experiments demonstrate the potential of our method for LTL synthesis, using LTL-to-Rabin translators. It leads to significantly smaller parity automata when compared to state-of-the-art approaches on complex formulae.

1 Introduction

Constructing correct-by-design systems from specifications given in linear temporal logic (LTL) [Pnu77] is a classical problem [PR89], called *LTL synthesis*. The automata-theoretic solution to this problem is to translate the LTL formula to a deterministic automaton and solve the corresponding game on the automaton. Although different kinds of automata can be used, a reasonable choice would be parity automata (DPA) due to the practical efficiency of parity game solvers [FL09,ML16] and the fact they allow for optimal memoryless strategies. The bottleneck is thus to create a reasonably small DPA. The classical way to transform LTL formulae into DPA is to first create a non-deterministic Büchi automaton (NBA) and then determinize it, as implemented in `ltl2dstar` [KB06]. Since determinization procedures [Pit06,Sch09] based on Safra's construction [Saf88] are practically inefficient, many alternative approaches to LTL synthesis arose, trying to avoid determinization and/or focusing on fragments of LTL, e.g. [KV05,PPS06,AL04]. However, new results on translating LTL directly and efficiently into deterministic automata [KE12,EK14] open new possibilities for the automata-theoretic approach. Indeed, tools such as Rabinizer [KK14] or LTL3DRA [BBKS13] can produce practically small deterministic Rabin automata (DRA). Consequently, the task is to efficiently transform DRA into DPA, which is the aim of this paper.

Transformations of deterministic automata into DPA are mostly based on *appearance records* [GH82]. For instance, for deterministic Muller automata, we

A. Legay and T. Margaria (Eds.): TACAS 2017, Part I, LNCS 10205, pp. 443–460, 2017.
DOI: 10.1007/978-3-662-54577-5_26

want to track which states appear infinitely often and which do not. In order to do that, the *state appearance record* keeps a permutation of the states, ordered according to their most recent visits, see e.g. [Sch01]. In contrast, for deterministic Streett automata (DSA) we only want to track which *sets* of states are visited infinitely often and which not. Consequently, *index appearance record* (IAR) keeps a permutation of these sets of interest instead, which are typically very few. Such a transformation has been given first in [Saf92] from DSA to DRA only (not DPA, which is a subclass of DRA). Fortunately, this construction can be further modified into a transformation of DSA to DPA, as shown in [Löd99b].

Since (1) DRA and DSA are syntactically the same, recognizing the complement languages of each other, and (2) DPA can be complemented without any cost, one can apply the IAR of [Löd99b] to DRA, too. However, we design another IAR, which is more natural from the DRA point of view, as opposed to the DSA perspective taken in [Löd99b]. This is in spirit more similar to a sketch of a construction suggested in [FEK11]. Surprisingly, we have found that the DRA perspective yields an algorithm producing considerably smaller automata than the DSA perspective.

Our contribution in this paper is as follows:

- We provide an IAR construction transforming DRA to DPA.
- We present optimizations applicable to all appearance records.
- We evaluate all the unoptimized and optimized versions of our IAR and the IAR of [Löd99b] experimentally, in comparison to the procedure implemented in GOAL [TTH13].
- We compare our approach LTL$\xrightarrow{\text{Rabinizer}}DRA\xrightarrow{\text{optimized IAR}}$DPA to the state-of-the-art translation of LTL to DPA by Spot 2.1 [DLLF+16], which mixes the construction of [Red12] with some optimizations of `ltl2dstar` [KB06] and of their own. The experiments show that for more complex formulae our method produces smaller automata.

2 Preliminaries on ω-automata

We recall basic definitions of ω-automata and establish some notation.

2.1 Alphabets and Words

An *alphabet* is any finite set Σ. The elements of Σ are called *letters*. A *word* is a (possibly infinite) sequence of letters. The set of all infinite words is denoted by Σ^ω. A set of words $\mathcal{L} \subseteq \Sigma^\omega$ is called *(infinite) language*. The i-th letter of a word $w \in \Sigma^\omega$ is denoted by w_i, i.e. $w = w_0 w_1 \dots$.

2.2 Transition Systems

A *deterministic transition system* (DTS) $\mathcal{T}$ is given by a tuple (Q, Σ, δ, q_0) where Q is a set of states, Σ is an alphabet, δ is a *transition function* $\delta : Q \times \Sigma \to Q$ which may be partial (due to technical reasons) and $q_0 \in Q$ is the *initial state*.

The transition function induces the *set of transitions* $\Delta = \{\langle q, a, q' \rangle \mid q \in Q, a \in \Sigma, q' = \delta(p, a)\}$. For a transition $t = \langle q, a, q' \rangle \in \Delta$ we say that t *starts at* q, *moves under* a and *ends in* q'. A sequence of transitions ρ is a *run* of a DTS $\mathcal{T}$ on a word $w \in \Sigma^\omega$ if ρ_0 starts at q_0, ρ_i moves under w_i for each $i \geq 0$ and ρ_{i+1} starts at the same state as ρ_i ends for each $i \geq 0$. We write $\mathcal{T}(w)$ to denote the unique run of $\mathcal{T}$ on w, if it exists. A transition t *occurs* in ρ if there is some i with $\rho_i = t$. By $\mathrm{Inf}(\rho)$ we denote the set of all transitions occurring infinitely often in ρ. Additionally, we extend Inf to words by defining $\mathrm{Inf}_{\mathcal{T}}(w) = \mathrm{Inf}(\mathcal{T}(w))$ if $\mathcal{T}$ has a run on w. If $\mathcal{T}$ is clear from the context, we write $\mathrm{Inf}(w)$ for $\mathrm{Inf}_{\mathcal{T}}(w)$.

2.3 Acceptance Conditions and ω-automata

An *acceptance condition* for $\mathcal{T}$ is a positive Boolean formula over the formal variables $V_\Delta = \{\mathrm{Inf}(T), \mathrm{Fin}(T) \mid T \subseteq \Delta\}$. Acceptance conditions are interpreted over runs as follows. Given a run ρ of $\mathcal{T}$ and such an acceptance condition α, we consider the truth assignment that sets the variable $\mathrm{Inf}(T)$ to true iff ρ visits (some transition of) T infinitely often, i.e. $\mathrm{Inf}(\rho) \cap T \neq \emptyset$. Dually, $\mathrm{Fin}(T)$ is set to true iff ρ visits every transition in T finitely often, i.e. $\mathrm{Inf}(\rho) \cap T = \emptyset$. A run ρ satisfies α if this truth-assignment evaluates α to true.

A *deterministic ω-automaton* over Σ is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, \alpha)$, where (Q, Σ, δ, q_0) is a DTS and α is an acceptance condition for it. An automaton $\mathcal{A}$ *accepts* a word $w \in \Sigma^\omega$ if the run of the automaton on w satisfies α. The language of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$. An acceptance condition α is a

- *Rabin condition* $\{(F_i, I_i)\}_{i=1}^k$ if $\alpha = \bigvee_{i=1}^k (\mathrm{Fin}(F_i) \wedge \mathrm{Inf}(I_i))$. Each (F_i, I_i) is called a *Rabin pair*, where the F_i and I_i are called the *prohibited set* and the *required set* respectively.
- *generalized Rabin condition* $\{(F_i, \{I_i^j\}_{j=1}^{k_i})\}_{i=1}^k$ if the acceptance condition is of the form $\alpha = \bigvee_{i=1}^n (\mathrm{Fin}(F_i) \wedge \bigwedge_{j=1}^{k_i} \mathrm{Inf}(I_j^k))$. This generalizes the Rabin condition, where each $k_i = 1$. Furthermore, every generalized Rabin automaton can be de-generalized into an equivalent Rabin automaton, which however may incur an exponential blow-up [KE12].
- *Streett condition* $\{(F_i, I_i)\}_{i=1}^k$ if $\alpha = \bigwedge_{i=1}^k (\mathrm{Inf}(F_i) \vee \mathrm{Fin}(I_i))$. Note that the Streett condition is exactly the negation of the Rabin condition and thus an automaton with a Rabin condition can be interpreted as a Streett automaton recognizing exactly the complement language.
- *Rabin chain condition* $\{(F_i, I_i)\}_{i=1}^k$ if it is a Rabin condition and $F_1 \subseteq I_1 \subseteq \cdots \subseteq F_k \subseteq I_k$. A Rabin chain condition is equivalent to a *parity condition*, specified by a priority assignment $\lambda : \Delta \to \mathbb{N}$. Such a parity condition is satisfied by a run ρ iff the maximum priority of all infinitely often visited transitions $\max\{\lambda(q) \mid q \in \mathrm{Inf}(\rho)\}$ is even.

A deterministic Rabin, generalized Rabin, Street or parity automaton is a deterministic ω-automaton with an acceptance condition of the corresponding kind. In the rest of the paper we use the corresponding abbreviations DRA, DGRA, DSA and DPA.

Furthermore, given a DRA with an acceptance set $\{(F_i, I_i)\}_{i=1}^k$ and a word $w \in \Sigma^\omega$, we write $\mathcal{F}_{\text{inf}} = \{F_i \mid F_i \cap \text{Inf}(w) \neq \emptyset\}$ and $\mathcal{I}_{\text{inf}} = \{I_i \mid I_i \cap \text{Inf}(w) \neq \emptyset\}$ to denote the set of all infinitely often visited prohibited and required sets, respectively.

3 Index Appearance Record

In order to translate (state-based acceptance) Muller automata to parity automata, a construction called *latest appearance record* has been devised[1]. In essence, the constructed state space consists of permutations of all states in the original automaton. In each transition, the state which has just been visited is moved to the front of the permutation. From this, one can deduce the set of all infinitely often visited states by investigating which states change their position in the permutation infinitely often along the run of the word. Such a constraint can be encoded as parity condition.

However, this approach comes with a very fast growing state space, as the amount of permutations grows exponentially. Moreover, applying this idea to transition based acceptance leads to even faster growth, as there usually are a lot more transitions than states. In contrast to Muller automata, the exact set of infinitely often visited transitions is not needed to decide acceptance of a word by a Rabin automaton. It is sufficient to know which of the prohibited and required *sets* are visited infinitely often. Hence, *index appearance record* uses the indices of the Rabin pairs instead of particular states in the permutation construction. This provides enough information to decide acceptance.

We introduce some formalities regarding permutations: For a given $n \in \mathbb{N}$, we use Π^n to denote the set of all permutations of $N = \{1, \ldots, n\}$, i.e. the set of all bijective functions $\pi : N \to N$. We identify π with its canonical representation as a vector $(\pi(1), \ldots, \pi(n))$. In the following, we will often say "the position of F_i in π" or similar to refer to the position of i in a particular π, i.e. $\pi^{-1}(i)$. With this, we define our variant of the index appearance record construction. Note that in contrast to previous constructions, ours is transition based, which also has a positive effect on the size of the produced automata, as discussed in our experimental results.

Definition 1 (Transition-based index appearance record for Rabin automata). *Let $\mathcal{R} = (Q, \Sigma, \delta, q_0, \{(F_i, I_i)\}_{i=1}^k)$ be a Rabin automaton. Then the* index appearance record automaton $\text{IAR}(\mathcal{R}) = (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{q}_0, \lambda)$ *is defined as the parity automaton with*

- $\tilde{Q} = Q \times \Pi^k$.
- $\tilde{q}_0 = (q_0, (1, \ldots, k))$.
- $\tilde{\delta}((q, \pi), a) = (\delta(q, a), \pi')$ *where π' is the permutation obtained from π by moving all indices of* prohibited *sets visited by the transition $t = \langle q, a, \delta(q, a)\rangle$*

[1] Originally, it appeared in an unpublished report of McNaughton under the name "order vector with hit".

to the front. Formally, let $\mathsf{Move} = \{i \mid t \in F_{\pi(i)}\}$ *be the set of positions of currently visited prohibited sets. If* $\mathsf{Move} = \emptyset$*, define* $\pi' = \pi$*, otherwise let* $n = |\mathsf{Move}|$ *and* $\mathsf{Move} = \{i_1, \ldots, i_n\}$*. With this*

$$\pi'(j) = \begin{cases} i_j & \text{if } j \leq n \\ \pi(j - n + |\{i \in \mathsf{Move} \mid i \leq j\}|) & \text{otherwise.} \end{cases}$$

– *To define the priority assignment, we first introduce some auxiliary notation. For a transition* $\tilde{t} = \langle (q, \pi), a, (q', \pi') \rangle$ *and its corresponding transition* $\langle q, a, q' \rangle$ *in the original automaton, let*

$$\mathrm{maxInd}(\tilde{t}) = \max(\{\pi^{-1}(i) \mid t \in F_i \cup I_i\} \cup \{0\})$$

be the maximal position of acceptance pair *in* π *visited by* t *(or* 0 *if none is visited). Using this, define the priority assignment as follows:*

$$\lambda(\tilde{t}) := \begin{cases} 1 & \text{if } \mathrm{maxInd}(\tilde{t}) = 0, \\ 2 \cdot \mathrm{maxInd}(\tilde{t}) & \text{if } t \in I_{\pi(\mathrm{maxInd}(\tilde{t}))} \setminus F_{\pi(\mathrm{maxInd}(\tilde{t}))} \\ 2 \cdot \mathrm{maxInd}(\tilde{t}) + 1 & \text{otherwise, i.e. if } t \in F_{\pi(\mathrm{maxInd}(\tilde{t}))}. \end{cases}$$

When a transition visits multiple prohibited sets, they can be moved to the front of the appearance record in arbitrary order. As an optimization we choose existing states as successors whenever possible.

Before formally proving correctness, i.e. that $\mathrm{IAR}(\mathcal{R})$ recognizes the same language as $\mathcal{R}$, we provide a small example in Fig. 1 and explain the general intuition behind the construction. For a given run, all prohibited sets which are visited infinitely often will eventually be "in front" of all those only seen finitely often: After some finite number of steps, none of the finitely often visited ones will be seen any more. Taking another sufficiently large amount of steps, every infinitely often visited set has been seen again and all their indices have been moved to the front.

Lemma 1. *Let* $w \in \Sigma^\omega$ *be a word on which* $\mathrm{IAR}(\mathcal{R})$ *has a run* $\tilde{\rho}$*. Then, the positions of all finitely often visited prohibited sets stabilize after a finite number of steps, i.e. their positions are identical in all infinitely often visited states. Moreover, for any* i, j *with* $F_i \in \mathcal{F}_{\mathrm{inf}}$*,* $F_j \notin \mathcal{F}_{\mathrm{inf}}$ *we have that the position of* F_i *is smaller than the position of* F_j *in every infinitely often visited state.*

Proof. The position of any F_i only changes in two different ways:

– Either F_i itself has been visited and thus is moved to the front,
– or some $F_{i'}$ with a position greater than the one of F_i has been visited and is moved to the front, increasing the position of F_i.

Let ρ be the run of $\mathcal{R}$ on w. (We prove the existence of such a run in [KMWW17, Lemma 3].) Assume that F_i is visited finitely often in some run ρ, i.e. there is a step in the run from which on F_i is never visited again. As the amount of

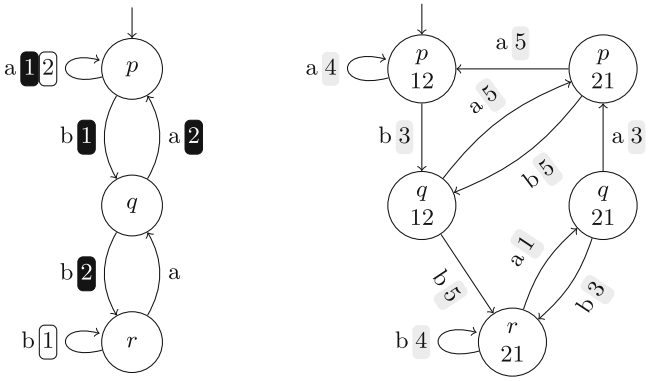


Fig. 1. An example DRA and its resulting IAR DPA. For the Rabin automaton, a number in a white box next to a transition indicates that this transition is a required one of that Rabin pair. A black shape dually indicates membership in the corresponding prohibited set. For example, with $t = \langle p, a, p \rangle$ we have $t \in F_1$ and $t \in I_2$. In the IAR construction, we shorten the notation for permutations to save space, so $p, 12$ corresponds to $(p, (1, 2))$. The priority of a transition is written next to the transitions letter.

positions is bounded, the second case may only occur finitely often after this step and the position of F_i eventually remains constant. As F_i was chosen arbitrarily, we conclude that all finitely often visited F_i are eventually moved to the right and remain on their position. Trivially, all infinitely often visited F_i move to the left, proving the claim. □

As an immediate consequence we see that if some transition $(q, a, q') \in F_i$ is visited infinitely often, then every F_j with a smaller position than F_i in q is also visited infinitely often:

Corollary 1. *Let $\tilde{t} \in \mathrm{Inf}_{\mathrm{IAR}(\mathcal{R})}(w)$ be an infinitely often visited transition with its corresponding transition $t \in F_{\pi(i)}$ for some i. Then $\forall j \leq i . F_{\pi(j)} \in \mathcal{F}_{\mathrm{inf}}$.*

Looking back at the definition of the priority function, the central idea of correctness can be outlined as follows. For every I_i which is visited infinitely often we can distinguish two cases:

- F_i is visited finitely often. Then the position of the pair is greater than the one of every $F_j \in \mathcal{F}_{\mathrm{inf}}$. Hence the priority of every transition $\tilde{t}$ with corresponding transition $t \in I_i$ is both even and bigger than every odd priority seen infinitely often along the run.
- F_i is visited infinitely often, i.e. after each visit of I_i, F_i is eventually visited. As argued in the proof of Lemma 1, the position of F_i can only increase until it is visited again. Hence every visit of I_i which yields an even parity is followed by a visit of F_i yielding an odd parity which is strictly greater.

Using this intuition, we formally show correctness of the construction in [KMWW17, Appendix A.1].

Theorem 1. *For any DRA $\mathcal{R}$ we have that $\mathcal{L}(\mathrm{IAR}(\mathcal{R})) = \mathcal{L}(\mathcal{R})$.*

Proposition 1 (Complexity). *For every DRA $\mathcal{R}$ with n states and k Rabin pairs, the constructed automaton $\mathrm{IAR}(\mathcal{R})$ has at most $n \cdot k!$ states and $2k + 1$ priorities.*

Moreover, using the [Löd99a], one can show that this is essentially optimal. There exists a family $\{\mathcal{L}_n\}_{n\geq 2}$ of languages such that for every n the language L_n can be recognized by a DRA with $O(n)$ states and $O(n)$ pairs, but cannot be recognized by a DPA with less than $n!$ states. For details, see [KMWW17, Appendix A.2].

Remark 1 (Comparison to previous IAR). Our construction is similar to the index appearance record of [Löd99b] in that it keeps the information about the current state and a permutation of pairs, implementing the appearance record. However, from the point of view of Streett automata, it is very natural to keep two pointers into the permutation, indicating the currently extreme positions of both types of sets in the accpetance condition. Indeed, this way we can keep track of all conjuncts of the form $\mathrm{Inf}(I_j) \implies \mathrm{Inf}(F_j)$. This is also the approach that [Löd99b] takes. In contrast, we have no pointers at all. From the Rabin point of view, it is more natural to keep track of the prohibited sets only and the respective pointer is hidden in the information about the current state *together* with the current permutation. Additionally, the pointer for the required set is hidden into the acceptance status of transitions. In the transition-based setting, it is not necessary to remember the visit of a required set in the state-space; it is sufficient to emit the respective priority upon seeing this *during* the transition when we know both the source and target states. The absence of these pointers results in better performance.

Remark 2 (Using IAR for DGRA). The straightforward way to translate a DGRA to DPA is to first de-generalize the DGRA and then apply the presented IAR construction. However, one can also apply the IAR idea to directly translate from DGRA to DPA: Instead of only tracking the pair indices, one could incorporate all F_i and I_i^j into the appearance permutation. With the same reasoning as above, a parity condition can be used to decide acceptance.

This approach yields a correct algorithm, but compared to de-generalization combined with IAR, the state space grows much larger. Indeed, given a DGRA with n states and k accepting pairs with l_i required sets each, the de-generalized DRA has at most $n \cdot \prod_{i=1}^{k} l_i$ states and k pairs, hence the resulting parity automaton has at most $k! \cdot n \cdot \prod_{i=1}^{k} l_i$ states and $2k+1$ priorities. Applying the mentioned specific construction gives $n \cdot (\sum_{i=1}^{k}(l_i+1))!$ states and $2 \cdot (\sum_{i=1}^{k}(l_i+1)) + 1$ priorities. A simple induction on k suffices to show that the worst case upper bound for the specific construction is always larger. We conjecture that this behaviour also shows in real-world applications.

4 Optimizations

In general, many states generated by the IAR procedure are often superfluous and could be omitted. In the following, we present several optimizations of our construction, which aim to do so. Moreover, these optimizations can be applied also to the IAR construction of [Löd99b] and in a slighly adjusted way also to the standard SAR [Sch01]. Further, although the optimizations are transition-based, they can be of course easily adapted to the state-based setting. Due to space constraints, the correctness proofs can be found in [KMWW17, Appendix A.3].

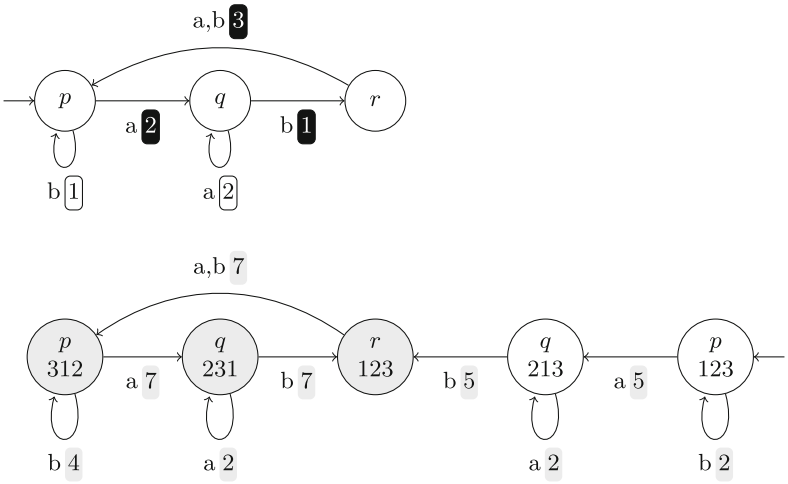


Fig. 2. Example of a suboptimal initial permutation, using the same notation as in Fig. 1. Only the shaded states are constructed when choosing a better initial permutation.

4.1 Choosing an Initial Permutation

The first observation is that the arbitrary choice of $(1, \ldots, k)$ as initial permutation can lead to suboptimal results. It may happen that several states of the resulting automaton are visited at most once by every run before some "recurrent" permutation is reached. These states enlarge the state-space unnecessarily, as demonstrated in Fig. 2. Indeed, when choosing $(p, (3, 1, 2))$ instead of $(p, (1, 2, 3))$ as the initial state in the example, only the shaded states are built during the construction, while the language of the resulting automaton is still equal to that of the input DRA.

We overload the IAR algorithm to be parametrized by the starting permutation, i.e. we write $\mathrm{IAR}(\mathcal{R}, \pi_0)$ to denote the IAR construction applied to the DRA $\mathcal{R}$ starting with permutation π_0.

Theorem 2. *For an arbitrary Rabin automaton $\mathcal{R}$ with k pairs we have that $\mathcal{L}(\mathrm{IAR}(\mathcal{R})) = \mathcal{L}(\mathrm{IAR}(\mathcal{R}, \pi_0))$ for all $\pi_0 \in \Pi^k$.*

How to choose a "good" initial permutation is deferred to Sect. 4.3, as it is intertwined with the algorithm presented in the following section.

4.2 SCC Decomposition

Acceptance of a word by an ω-automaton only depends on the set of states visited infinitely often by its run. This set of states is *strongly connected* on the underlying graph structure, i.e. starting from any state in the set, any other state can be reached with finitely many steps. In general, any strongly connected set belongs to exactly one *strongly connected component* (SCC). Therefore, for a fixed SCC, only the Rabin pairs with required sets intersecting this SCC are relevant.

Using this we can restrict ourselves to the Rabin pairs that can possibly accept in that SCC while processing it. This reduces the number of indices we need to track in the appearance record for each SCC, which can lead to significant savings.

For readability, we introduce some abbreviations. Given a DRA $\mathcal{R} = (Q, \Sigma, \delta, q_0, \{(F_i, I_i)\}_{j=1}^k)$ and a set of states $S \subseteq Q$ we write $\delta \restriction S : S \times \Sigma \to S$ to denote the restriction of δ to S, i.e. $\delta \restriction S(q, a) = \delta(q, a)$ if $\delta(q, a) \in S$ and undefined otherwise. Analogously, we define $\Delta \restriction S = \Delta \cap S \times \Sigma \times S$ as the set of transitions in the restricted automaton. Consequently, we define the restriction of the whole automaton $\mathcal{R}$ to the set of states S using $q \in S$ as initial state by

$$\mathcal{R} \restriction_q S = (S, \Sigma, \delta \restriction S, q, \{(F_i \cap (\Delta \restriction S), I_i \cap (\Delta \restriction S)) \mid I_i \cap (\Delta \restriction S) \neq \emptyset\}).$$

Furthermore, we call a SCC of an automaton *transient*, if it is a singleton set without a self-loop. This means that it is visited at most once by any run and it is not of interest for acceptance. Finally, we use ε to denote the "empty" permutation (of length 0).

Using this notation, we describe the optimized IAR construction, denoted IAR* in Algorithm 1. The algorithm decomposes the DRA into its SCCs, applies the formerly introduced IAR procedure to each sub-automaton separately and finally connects the resulting DPAs back together.

As we apply the IAR construction to each SCC separately, we have to choose the initial permutation for each state of those SCCs. Theorem 2 shows that for a particular initial state, correctness of IAR does not depend on the chosen permutation. We therefore delegate the choice to a function pickPerm and prove correctness of the optimized algorithm independent of this function, allowing for further optimizations. We present an optimal definition of pickPerm in the next subsection.

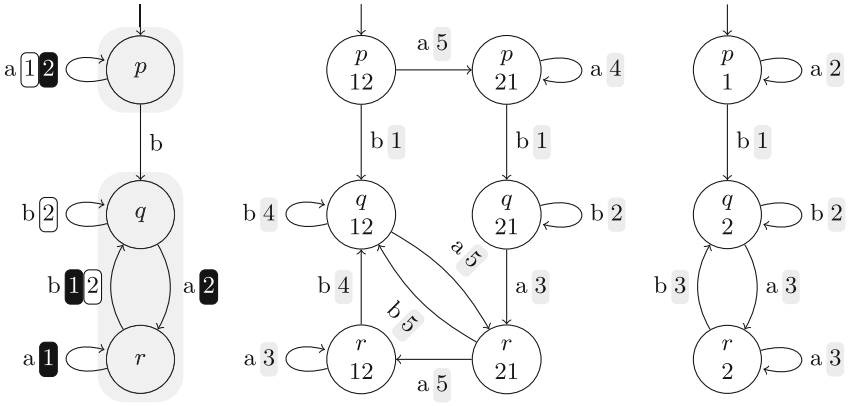
Figure 3 shows an example application and the obtained savings of the construction. Pair 1 is only relevant for acceptance in the SCC $\{p\}$, but in the unoptimized construction it still changes the permutations in the part of the

Input : A DRA $\mathcal{R} = (Q, \Sigma, \delta, q_0, \{(F_i, I_i)\}_{j=1}^{k})$

Output: A DPA recognizing the same language as $\mathcal{R}$

$Q^* \leftarrow \{\}$, $\delta^* \leftarrow \{\}$, $\lambda^* \equiv 1$
foreach *SCC* S *in* $\mathcal{R}$ **do**
  if S *transient or* $\{i \mid I_i \cap \Delta \restriction S \neq \emptyset\} = \emptyset$ **then** `// SCC not relevant`
    Add $S \times \{\varepsilon\}$ to Q^*
    foreach $q \in S$, $a \in \Sigma$ *such that* $(\delta \restriction S)(q, a)$ *is defined* **do**
      Let $q' = \delta(q, a)$
      Set $\delta^*((q, \varepsilon), a) = (q', \varepsilon)$ and $\lambda^*(\langle (q, \varepsilon), a, (q', \varepsilon) \rangle) = 1$
    end
  else `// SCC relevant, apply IAR to the sub-automaton`
    Pick a starting state $q \in S$
    $(Q_S, \Sigma, \delta_S, (q, \pi), \lambda_S) \leftarrow \mathrm{IAR}(\mathcal{R} \restriction_q S, \textsc{pickPerm}(q, S))$
    Update Q^*, δ^* and λ^* with Q_S, δ_S and λ_S, respectively
  end
end
`// Connect all SCCs`
foreach $(q, \pi) \in Q^*$ *and* $a \in \Sigma$ *s.t.* $q' = \delta(q, a)$ *in different SCC of* $\mathcal{R}$ *than* q **do**
  Pick a π' with $(q', \pi') \in Q^*$
  Set $\delta^*((q, \pi), a) = (q', \pi')$
end

Algorithm 1. The optimized IAR construction IAR*



Example DRA, SCCs shaded

Result of unoptimized IAR applied to the DRA

Output of the optimized IAR*.

Fig. 3. Example application of Algorithm 1

automaton constructed from $\{q, r\}$, as e.g. the transition $\langle r, b, q \rangle$ is contained in F_1. Similarly, pair 2 is tracked in $\{p\}$ while actually not being relevant. The optimized version yields improvements in both state-space size and amount of priorities.

Theorem 3. *For any DRA $\mathcal{R}$ we have that $\mathcal{L}(\mathrm{IAR}^*(\mathcal{R})) = \mathcal{L}(\mathcal{R})$, independent of* pickPerm.

4.3 Optimal Choice of the Initial Permutation

In Fig. 2 we provided a scalable example where the choice of the initial permutation can significantly reduce the size of the generated automaton. In this subsection, we explain a procedure yielding a permutation which minimizes the state space of the automaton generated by IAR*.

First, we recall that pickPerm is only invoked when processing a particular (non-transient) SCC of the input automaton. Consequently, we can restrict ourselves to only deal with Rabin automata forming a single SCC. Let now $\mathcal{R}$ be such an automaton. While $\mathrm{IAR}(\mathcal{R}, \pi_0)$ may contain multiple SCCs, we show that it contains exactly one bottom SCC (BSCC), i.e. a SCC without outgoing edges. Additionally, this BSCC is the only SCC which contains all states of the original automaton $\mathcal{R}$ in the first component of its states.

Theorem 4. *Let $\mathcal{R} = (Q, \Sigma, \delta, q_0, \{(F_i, I_i)\}_{i=1}^k)$ be a Rabin automaton that is strongly connected. For a fixed $\pi_0 \in \Pi^k$, $\mathrm{IAR}(\mathcal{R}, \pi_0)$ contains exactly one BSCC S and for every SCC S' we have that $S = S'$ iff $Q = \{q \mid \exists \pi \in \Pi^k.(q, \pi) \in S'\}$. Furthermore the BSCCs for different π_0 are isomorphic.*

The proof can be found in [KMWW17, Appendix A.4]. This result makes defining an optimal choice of pickPerm straightforward. By the theorem, there always is a BSCC of the same size, independent of pickPerm. If (q_0, π) is in the BSCC of some $\mathrm{IAR}(\mathcal{R}, \pi_0)$, $\mathrm{IAR}(\mathcal{R}, \pi)$ will generate the same BSCC and no other states. Hence, we define $\textsc{pickPerm}(q, S)$ to return any permutation such that (q, π) lies in the corresponding BSCC. As a trivial consequence of the theorem, this choice is optimal in terms of the state-space size of the generated automaton. In our implementation, we start exploring the state space using an arbitrary initial permutation and then prune all states which do not belong into the respective BSCC.

5 Experimental Results

In this section, we compare variants of our new approach to the established tools. All of the benchmarks have been run on a Linux 4.4.3-gentoo x64 virtual machine with 3.0 GHz per core. We implemented our construction as part of Rabinizer [KK14] and used the 64 bit Oracle JDK 1.8.0_102 as JVM for our experiments.

5.1 DRA to DPA Translation

We present comparisons of different approaches to translate DRA into DPA. As there are to our knowledge no "standard" DRA datasets for this kind of

comparison, we use Spot's tool `randaut` to produce various Rabin automata. All executions in this chapter ran with a time-out of five minutes.

We consider both our basic method IAR of Sect. 3 and the optimized version IAR* of Sect. 4. We compare our methods to GOAL[2] [TTH13] and the Streett-based construction StreetIAR of [Löd99b]. As we are not aware of any implementations of StreetIAR, we implemented it ourselves in Haskell[3]. Both of these constructions are using state-based acceptance. In order to allow for a fair comparison, we therefore also implemented sbIAR, a variant of our construction working directly with state-based acceptance[4] in Haskell (See footnote 3), too. Additionally, we combine every tool with Spot's multi-purpose post-processing[5] and denote this by a subscript P (for post-processing), e.g. IAR* combined with this post-processing is written IAR^*_P.

In Table 1 we present a comparison between GOAL, StreettIAR and our unoptimized state-based implementation sbIAR. Additionally, since GOAL does not perform too well, we also include its post-processed variant GOAL_P. For comparison, we also include our optimized variant IAR^*_P. As test data, we use 1000 state-based DRA over 4 atomic propositions with 5 to 15 states, a transition density of 0.05 and 2 to 3 Rabin pairs[6]. We use Spot's tool `autfilt` to gather the statistics. Failures denote either time-outs, out of memory errors or invalid results, e.g. automata which could not be read by `autfilt`, which sometimes occurred with GOAL.

Table 1. Comparison of the DRA to DPA translations on 1000 randomly generated DRAs. First, we compare the cases where all tools finished successfully, according to the average size, the number of SCCs and the run-time. Second, we give the percentage each tool produces an automaton with the least number of states, and failures, respectively.

		GOAL	GOAL_P	StreettIAR	sbIAR	IAR^*_P
Avg.	#states	1054	281	18.4	15.4	8.83
	#SCC	73.2	19.2	4.97	4.33	1.61
	time (s)	11.7	15.7	0.02	0.02	0.99
Smallest (%)		15.5	37.8	7.7	15.5	95.9
Failure (%)		8.6	11.9	0	0	0

[2] `gc batch"\$nba = load -c HOAF /dev/stdin; \$dpa = convert -t dpw \$nba; save \$dpa -c HOAF /dev/stdout;"`, executed with `OpenJDK IcedTea 2.6.6, java version 1.7.0_101`.

[3] Compiled with GHC 7.10.3.

[4] We also proved correctness for the direct construction, the proof can be obtained by trivial modifications of the proofs in this paper.

[5] `autfilt --deterministic --generic --small --high`.

[6] `randaut 4 --seed=0 -Q 5..15 --acceptance="Rabin 2..3"--density=0.05 --deterministic --acc-probability 0.2 --state-based-acceptance --hoaf -n1000`. The acceptance probability parameter denotes the chance of a particular transition belonging to a Rabin pair.

From the results in Table 1 we observe that on this dataset all appearance-record variants drastically outperform GOAL. We remark that IAR* performs even better compared to GOAL if more SCCs are involved. However, for reasonably complex automata, virtually every execution of GOAL timed out or crashed, making more specific experiments difficult. Already for the automata in Table 1 with 5–15 states, GOAL regularly consumed around 3 GB of memory and needed roughly 10 seconds to complete on average, whereas our methods only used a few hundred MB and less than a second. We could not find a dataset where GOAL showed a significant advantage over our new methods. Therefore, we exclude GOAL from further experiments. The remaining methods are investigated more thoroughly in the next experiment.

Table 2. Comparison of StreettIAR and (sb)IAR on 1000 randomly generated DRAs. We use the same definitions as in Table 1.

		StreettIAR	sbIAR	StreettIAR$^*_P$	sbIAR$^*_P$	IAR$^*_P$
Avg.	#states	4959	1568	4175	1081	833
	#SCC	63.8	42.5	1.35	1.35	1.35
	time (s)	1.86	0.34	39.47	3.11	3.38
Smallest (%)		0	0	0.4	5.90	95.1
Failure (%)		1.3	0	1.4	0	0

In Table 2 we compare StreettIAR to sbIAR on more complex input automata to demonstrate the advantages of our new method compared to the existing StreettIAR construction. We consider the methods both in the basic setting and with post-processing and optimizations. Note that as the presented optimizations are applicable to appearance records in general, we also added them to our implementation of StreettIAR. Its optimized version is denoted by StreettIAR*. Again, we include our best (transition-based) variant IAR$^*_P$ for reference. The dataset now contains DRA with 20 to 30 states[7].

StreettIAR is significantly outperformed by our new methods in this experiment. This is quite surprising, considering that both methods essentially follow the same idea of index appearance records, only from different perspectives. The difference is partially due to Remark 1. Besides, we have observed that the discrepancy between StreettIAR and IAR is closely linked to the amount of acceptance pairs. After increasing the number of pairs further, the gap between the two approaches grows dramatically. For instance, on a dataset of automata with 8 states and 8 Rabin pairs, the IAR construction yielded automata roughly an order of magnitude smaller: sbIAR needed less than three hundred states

[7] `randaut 4 --seed=0 -Q 20..30 --acceptance="Rabin 6"--density=0.05 --acc-probability=0.2 --deterministic --state-based-acceptance --hoaf-n1000`.

compared to StreettIAR needing over three thousand. Applying the post-processing does not remedy the situation.

Table 3. Evaluation of the presented optimizations on 1000 randomly generated DRAs, again using the same definitions as in Table 1. No tool failed for any of the input automata.

		sbIAR	sbIAR^*_P	IAR	IAR_P	IAR^*	IAR^*_P
Avg.	#states	3431	2530	1668	1655	1302	1296
	#SCC	24.8	1.14	8.98	3.5	1.43	1.43
	Time (s)	0.77	11.47	1.09	48.3	76.5	95.7
Smallest (%)		0	0	38.3	48.30	76.5	95.7

Finally, we demonstrate the significance of the transition-based acceptance and our optimizations in Table 3. To evaluate the impact of our improvements, we compare the unoptimized IAR procedure and its post-processed counterpart to the optimized IAR^* and IAR^*_P. Furthermore, we also include our state-based version in its basic (sbIAR) and best (sIAR^*_P[8]) form. We run these algorithms on a dataset of DRA with 20 states each[9].

Spot's generic post-processing algorithms often yield sizeable gains, but they are marginal compared to the effect of our optimizations on this dataset. Our optimizations are thus not only significantly beneficial, but also irreplacable by general purpose optimizations. We furthermore want to highlight the reduction of SCCs. As a final remark, we emphasize the improvements due to the adoption of transition-based acceptance, halving the size of the automata.

5.2 Linear Temporal Logic

Motivated by the previous results we concatenated IAR^* with Rabinizers LTL-to-DRA translation, obtaining an LTL-to-DPA translation. We compare this approach to the established tool `ltl2tgba` of Spot, which can also produce DPA[10]. We use Spot's comparison tool `ltlcross` in order to produce the results. Unfortunately, this tool sometimes crashes caused by too many acceptance sets[11]. We alleviated this problem by splitting our datasets into smaller chunks. Time-outs are set to 15 min.

[8] We use `autfilt --state-based-acceptance` to convert the transition based input DRA to state based.

[9] `randaut 4 --seed=0 -Q 20 --acceptance="Rabin 5"--acc-probability=0.05--density=0.1 --deterministic --hoaf -n1000`.

[10] By specifying `--deterministic --generic` on the command line.

[11] Around 20 acceptance sets. The exact error message emitted is `-terminate called after throwing an instance of'std::runtime_error' what(): Too many acceptance sets used.`

First, we compare the two tools on random LTL formulae. We use `randltl` and `ltlfilt` to generate pure LTL formulae[12]. The test results are outlined in Table 4. On average, our methods are comparable to `ltl2tgba`, even outperforming it slightly in the number of states.

Note that the averages have to be compared carefully. As the constructions used by `ltl2tgba` are fundamentally different from ours, there are some formulae where we outperform `ltl2tgba` by orders of magnitude and similarly in the other direction. We conjecture that on some formulae `ltl2tgba` has an edge merely due to its rewriting together with numerous pre- and post-processing steps, whereas our method profits from Rabinizer, which can produce smaller deterministic automata also for very complex formulae. On many dataset we tested, median state count over all formulae (including timeouts) is better for our methods. For more detail, see the histogram in [KMWW17, Appendix B, Fig. 4].

Table 4. Comparison of `ltl2tgba` to Rabinizer + IAR^*_P on 2000 LTL formulae.

		Rabinizer + IAR^*_P	`ltl2tgba`
Avg.	#states	6.60	7.89
	#acc	2.31	1.79
	#SCC	4.49	4.69
	Timeouts	22	0

To give more insight in the difference between the approaches, we list several classes of formulae where our technique performs particularly well. For instance, for fairness-like constraints our toolchain produces significantly smaller automata than `ltl2tgba`, see Table 5. Further examples, previously investigated in e.g. [KE12, BBKS13, EK14] can be found in [KMWW17, Appendix B, Table 6], including formulae of the GR(1) fragment [PPS06]. Additionally, our method is performing better on many practical formulae, for instance complex formulae from Spec Pattern [DAC99][13].

6 Conclusion

We have presented a new version of index appearance record. In comparison to the standard Streett-based approach, our new Rabin-based approach produces significantly smaller automata. Besides, it has a significant potential for LTL synthesis. For more complex formulae, it makes use of high efficiency of Rabinizer

[12] `randltl -n2000 5 --tree-size=20..25 --seed=0 --simplify=3 -p --ltl-priorities' ap=3, false=1,true=1,not=1,F=1,G=1,X=1,equiv=1,implies=1, xor=0,R=0,U=1,W=0,M=0,and=1,or=1' | ltlfilt --unabbreviate="eiMRW"`.

[13] Spec Patterns: Property Pattern Mappings for LTL. http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml.

Table 5. Fairness formulae: $Fairness(k) = \bigwedge_{i=1}^{k}(\mathbf{G}\,\mathbf{F}\,a_i \Rightarrow \mathbf{G}\,\mathbf{F}\,b_i)$

	Rabinizer+IAR$^*_P$			`ltl2tgba`		
Formula	States	Acc	SCCs	States	Acc	SCCs
Fairness(1)	2	4	1	5	4	3
Fairness(2)	12	9	1	44	8	9
Fairness(3)	1431	17	1	8607	20	546

and thus avoids the blow-up in many cases, compared to determinization-based methods.

Since we only provided the method for DRA we want to further investigate whether it can be extended to DGRA more efficiently than by de-generalization. Besides, a more targeted post-processing of the state space and the priority function is desirable. For instance, in order to decrease the total number of used priorities, all non-accepting SCCs can be assigned any odd priority that is already required elsewhere instead of the one suggested by the algorithm. Further, one can adopt optimizations of Spot as well as consider optimizations taking the automaton topology more into account. The whole tool-chain will then be integrated into Rabinizer. Finally, in order to estimate the effect on LTL synthesis more precisely, we shall link our tool chain to parity-game solvers and apply it to realistic case studies.

Acknowledgment. This work is partially funded by the DFG project "Verified Model Checkers" and by the Czech Science Foundation, grant No. P202/12/G061.

References

[AL04] Alur, R., La Torre, S.: Deterministic generators and games for LTL fragments. ACM Trans. Comput. Log. **5**(1), 1–25 (2004)

[BBKS13] Babiak, T., Blahoudek, F., Křetínský, M., Strejček, J.: Effective translation of LTL to deterministic Rabin automata: beyond the (F,G)-fragment. In: Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 24–39. Springer, Heidelberg (2013). doi:10.1007/978-3-319-02444-8_4

[DAC99] Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE, pp. 411–420 (1999)

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

[EK14] Esparza, J., Křetínský, J.: From LTL to deterministic automata: a safraless compositional approach. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 192–208. Springer, Heidelberg (2014). doi:10.1007/978-3-319-08867-9_13

[FEK11] Finkbeiner, B., Ehlers, R., Kupriyanov, A: Automata, games, and verification (2011). https://www.react.uni-saarland.de/teaching/automata-games-verification-11/downloads/ps9.pdf. Accessed 30 Aug 2016

[FL09] Friedmann, O., Lange, M.: Solving parity games in practice. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 182–196. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04761-9_15

[GH82] Gurevich, T., Harrington, L.: Trees, automata, and games. In: STOC, pp. 60–65 (1982)

[KB06] Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of linear temporal logic. Theoret. Comput. Sci. **363**(2), 182–195 (2006)

[KE12] Křetínský, J., Esparza, J.: Deterministic Automata for the (F,G)-fragment of LTL. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 7–22. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31424-7_7

[KK14] Komárková, Z., Křetínský, J.: Rabinizer 3: safraless translation of LTL to small deterministic automata. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 235–241. Springer, Heidelberg (2014). doi:10.1007/978-3-319-11936-6_17

[KMWW17] Křetínský, J., Meggendorfer, T., Waldmann, C., Weininger, M.: Index appearance record for transforming Rabin automata into parity automata. Technical report abs/1701.05738, arXiv.org (2017)

[KV05] Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In FOCS, pp. 531–542, (2005)

[Löd99a] Löding, C.: Optimal bounds for transformations of ω-automata. In: Rangan, C.P., Raman, V., Ramanujam, R. (eds.) FSTTCS 1999. LNCS, vol. 1738, pp. 97–109. Springer, Heidelberg (1999). doi:10.1007/3-540-46691-6_8

[Löd99b] Löding, C.: Methods for the transformation of automata: complexity and connection to second order logic. Master's thesis, Institute of Computer Science and Applied Mathematics, Christian-Albrechts-University of Kiel, Germany (1999)

[ML16] Meyer, P.J., Luttenberger, M.: Solving mean-payoff games on the GPU. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 262–267. Springer, Heidelberg (2016). doi:10.1007/978-3-319-46520-3_17

[Pit06] Piterman, N.: From nondeterministic Buchi and Streett automata to deterministic parity automata. In: LICS, pp. 255–264 (2006)

[Pnu77] Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57 (1977)

[PPS06] Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of Reactive(1) Designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2005). doi:10.1007/11609773_24

[PR89] Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL, pp. 179–190 (1989)

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

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

[Saf92] Safra, S.: Exponential determinization for omega-automata with strong-fairness acceptance condition (extended abstract). In: STOC, pp. 275–282 (1992)

[Sch01] Schwoon, S.: Determinization and complementation of Streett automata. In: Grädel, E., Thomas, W., Wilke, T. (eds.) Automata Logics, and Infinite Games. LNCS, vol. 2500, pp. 79–91. Springer, Heidelberg (2002). doi:10.1007/3-540-36387-4_5

[Sch09] Schewe, S.: Tighter bounds for the determinisation of Büchi automata. In: Alfaro, L. (ed.) FoSSaCS 2009. LNCS, vol. 5504, pp. 167–181. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00596-1_13

[TTH13] Tsai, M.-H., Tsay, Y.-K., Hwang, Y.-S.: GOAL for Games, Omega-Automata, and Logics. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 883–889. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_62