Tight Bounds for the Determinisation and Complementation of Generalised Büchi Automata*

Sven Schewe and Thomas Varghese

University of Liverpool

Abstract. Generalised Büchi automata are Büchi automata with multiple accepting sets. They form a class of automata that naturally occurs, e.g., in the translation from LTL to ω-automata. In this paper, we extend current determinisation techniques for Büchi automata to generalised Büchi automata and prove that our determinisation is optimal. We show how our optimal determinisation technique can be used as a foundation for complementation and establish that the resulting complementation is tight. Moreover, we show how this connects the optimal determinisation and complementation techniques for ordinary Büchi automata.

1 Introduction

While finite automata over infinite words were first introduced in Büchi's decidability proof for the monadic second-order logic of one successor (S1S) [1], they are most widely used in model checking, realisability checking, and synthesis procedures for linear time temporal logic (LTL) [11].

Büchi automata are an adaptation of finite automata to languages over infinite words. They differ from finite automata only with respect to their acceptance condition: while finite runs of finite automata are accepting if a final state is visited at the end of the run, an infinite run of a Büchi automaton is accepting if a final state is visited (or a final transition is taken) infinitely many times. Although this might seem to suggest that automata manipulations for Büchi automata are equally simple as those for finite automata, this is unfortunately not the case. In particular, Büchi automata are not closed under determinisation. While a simple subset construction suffices to efficiently determinise finite automata [13], deterministic Büchi automata are strictly less expressive than nondeterministic Büchi automata. For example, deterministic Büchi (or generalised Büchi) automata cannot recognise the simple ω-regular language that consists of all infinite words that contain only finitely many a's.

Determinisation therefore requires automata with more involved acceptance mechanisms [14, 9, 10, 4, 17], such as Muller's subset condition [8], Rabin's *pairs* condition [12] or its dual, the Streett condition [19], or the parity condition. Also, an $n^{\Omega(n)}$ lower bound for the determinisation of Büchi automata has been established [20] even if we allow for Muller objectives, which implies that a simple subset construction cannot suffice.

* This work was supported by the Engineering and Physical Science Research Council grant EP/H046623/1 'Synthesis and Verification in Markov Game Structures'.

Rabin's extension of the correspondence between automata and monadic logic to the case of trees [12] built on McNaughton's doubly exponential determinisation construction [8], and Muller and Schupp's [9] efficient nondeterminisation technique for alternating tree automata is closely linked to the determinisation of nondeterministic word automata. Safra was the first to introduce singly-exponential determinisation constructions for Büchi [14] and Streett [15] automata and current determinisation techniques [10, 17] build on Safra's work. For instance, Schewe's determinisation technique for Büchi automata [17] builds on Safra's [14] and Piterman's [10] determinisation procedures using a separation of concern, where the main acceptance mechanism, represented by *history trees*, is separated from the formal acceptance condition, e.g., a Rabin or parity condition. History trees can be seen as a simplification of Safra trees [14]. In a nutshell, they represent a family of breakpoint constructions; sufficiently many to identify an accepting run, and sufficiently few to be concise.

The standard translation from LTL to ω-automata [3] goes to *generalised* Büchi automata, which have multiple accepting sets and require that a final state from each accepting set is visited infinitely many time. There are several ways to determinise a generalised Büchi automaton with n states and k accepting sets. One could start with translating the resulting generalised Büchi automaton first to an ordinary nondeterministic Büchi automaton with nk states and a single accepting set, resulting in a determinisation complexity of roughly $(nk)^{O(nk)}$ states, or one could treat it as a Streett automaton, which is equally expensive and has a more complex determinisation construction.

Schewe's determinisation procedure [17] proves to be an easy target for generalisation, because it separates the acceptance mechanism from the accepting condition. To extend this technique from ordinary to generalised Büchi, it suffices to apply a round-robin approach to all breakpoints under consideration. That is, each subset is enriched by a natural number identifying the accepting set, for which we currently seek for the following breakpoint. Each time a breakpoint is reached, we turn to the next accepting set. Note that this algorithm is a generalisation in the narrower sense: in case that there is exactly one accepting set, it behaves exactly as the determinisation procedure for Büchi automata in [17]. An algorithm to determinise generalised Büchi automata to deterministic parity automata using this method was used in [5], similarly extending Piterman's construction [10, 7].

Using the current techniques and bounds for the determinisation of these automata [10, 17, 7], we find that for a nondeterministic generalised Büchi automaton with n states and k accepting sets, we get a deterministic Rabin automaton with $\mathsf{ghist}_k(n)$ states and $2^n - 1$ Rabin pairs. The function $\mathsf{ghist}_k(n)$ is approximately $(1.65\,n)^n$ for $k = 1$, $(3.13\,n)^n$ for $k = 2$, and $(4.62n)^n$ for $k = 3$, and converges against $(1.47kn)^n$ for large k. These bounds can also be used to establish smaller maximal sizes of minimal models, which is useful for Safraless determinisation procedures [6, 18, 5]. It would be simple to extend the transformation to deterministic parity automata from [17] to obtain an automaton with $O(n!^2\,2^k)$ states and $2n+1$ priorities. In this sense, the difference between determinising Büchi and generalised Büchi is negligible if k is small compared to n.

Colcombet and Zdanowski [2] showed that Schewe's determinisation procedure for Büchi automata is optimal. Our extension of this lower bound to generalised Büchi automata generalises their techniques, showing that the determinisation is optimal.

We also discuss a bridge between optimal determinisation and tight complementation. We show how the nondeterministic power of an automaton can be exploited by using a more concise data structure compared to determinisation (flat trees instead of general ones). This bridge again results in a generalisation of the Büchi complementation procedure discussed in [16] in the narrower sense: for one accepting set, the resulting automata coincide. We also provide a matching lower bound: we show for alphabets L_n^k that the size of a generalised Büchi automaton that recognises the complement of a full generalised Büchi automaton with n states and k accepting sets must be larger than $|L_n^k|$, while the ordinary Büchi automaton we construct is smaller than $|L_{n+1}^{k+1}|$. For large k – that is, if k is not small compared to n – $|L_n^k|$ is approximately $\left(\frac{kn}{e}\right)^n$. This improves significantly over the $\big(\Omega(nk)\big)^n$ bound established by Yan [20].

2 Nondeterministic and deterministic automata

Nondeterministic Rabin automata are used to represent ω-regular languages $L \subseteq \Sigma^\omega = \omega \to \Sigma$ over a finite alphabet Σ. In this paper, we use automata with trace based acceptance mechanisms. We denote $\{1,2,\ldots,k\}$ by $[k]$.

A nondeterministic Rabin automaton with k accepting pairs is a quintuple $\mathcal{A} = (\Sigma, Q, I, T, \{(A_i, R_i) \mid i \in [k]\})$, consisting of a finite alphabet Σ, a finite set Q of states with a non-empty subset $I \subseteq Q$ of initial states, a set $T \subseteq Q \times \Sigma \times Q$ of transitions from states through input letters to successor states, and a finite family $\{(A_i, R_i) \in 2^T \times 2^T \mid i \in [k]\}$ of Rabin pairs.

Nondeterministic Rabin automata are interpreted over infinite sequences $\alpha : \omega \to \Sigma$ of input letters. An infinite sequence $\rho : \omega \to Q$ of states of $\mathcal{A}$ is called a *run* of $\mathcal{A}$ on an input word α if the first letter $\rho(0) \in I$ of ρ is an initial state, and if, for all $i \in \omega$, $\big(\rho(i), \alpha(i), \rho(i+1)\big) \in T$ is a transition. For a run ρ of a word α, we denote with $\overline{\rho} : i \mapsto \big(\rho(i), \alpha(i), \rho(i+1)\big)$ the *transitions of* ρ.

A run $\rho : \omega \to Q$ is *accepting* if, for some index $i \in [k]$, some transition $t \in A_i$ in the accepting set A_i of the Rabin pair (A_i, R_i), but no transition $t' \in R_i$ from the rejecting set R_i of this Rabin pair appears infinitely often in the transitions of ρ, $\overline{\rho}$. $(\exists i \in [k].\ \mathit{inf}(\overline{\rho}) \cap A_i \neq \emptyset \wedge \mathit{inf}(\overline{\rho}) \cap R_i = \emptyset$ for $\mathit{inf}(\overline{\rho}) = \{t \in T \mid \forall i \in \omega\ \exists j > i$ such that $\overline{\rho}(j) = t\})$. A word $\alpha : \omega \to \Sigma$ is *accepted* by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run on α, and the set $\mathcal{L}(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \alpha$ is accepted by $\mathcal{A}\}$ of words accepted by $\mathcal{A}$ is called its *language*.

For technical convenience we also allow for finite runs $q_0 q_1 q_2 \ldots q_n$ with $T \cap \{q_n\} \times \{\alpha(n)\} \times Q = \emptyset$. Naturally, no finite run satisfies the Rabin condition. Finite runs are rejecting, and have no influence on the language of an automaton.

Two particularly simple types of Rabin automata are of special interest: generalised Büchi, and Büchi automata. Büchi automata are Rabin automata with a single accepting pair $(F, \emptyset)$ with an empty set of rejecting transitions. The transitions in F are traditionally called final, and Büchi automata are denoted $\mathcal{A} = (\Sigma, Q, I, T, F)$.

Generalised Büchi automata, denoted $\mathcal{A} = (\Sigma, Q, I, T, \{F_i \mid i \in [k]\})$, have a family of accepting (or: final) sets. A run ρ of a generalised Büchi automaton is accepting if it contains infinitely many transitions from all final sets ($\forall i \in [k].\ \mathit{inf}(\overline{\rho}) \cap F_i \neq \emptyset$).

Let $\delta : (q, \sigma) \mapsto \{q' \in Q \mid (q, \sigma, q') \in T\}$ denote the successor function for a set of transitions. A Rabin, Büchi, or generalised Büchi automaton is called *deterministic*

if δ is deterministic ($\forall(q,\sigma) \in Q \times \Sigma.\ |\delta(q,\sigma)| \leq 1$) and $I = \{q_0\}$ is singleton. We denote deterministic automata as $(\Sigma, Q, q_0, \delta', \Gamma)$, where Γ is the acceptance condition, and $\delta' : Q \times \Sigma \to Q$ is the partial function that is undefined in the preimage of $\emptyset$ for δ and otherwise satisfies $\delta(q,\sigma) = \{\delta'(g,\sigma)\}$.

3 Determinisation

The determinisation construction described in this section is a generalisation of Schewe's construction for nondeterministic Büchi automata [17], which in turn is a variation of Safra's [14]. We first define the structure that captures the acceptance mechanism of our deterministic Rabin automaton.

Ordered trees. We call a tree $\mathcal{T} \subseteq \mathbb{N}^*$ an *ordered tree* if it satisfies the following constraints, and use the following terms:

- every element $v \in \mathcal{T}$ is called a *node*,
- if a node $v = n_1 \dots n_j n_{j+1} \in \mathcal{T}$ is in $\mathcal{T}$, then $v' = n_1 \dots n_j$ is also in $\mathcal{T}$, v' is called the *predecessor* of v, denoted $\mathsf{pred}(v)$ ($\mathsf{pred}(\varepsilon)$ is undefined),
- the empty sequence $\varepsilon \in \mathcal{T}$, called the *root*, is in $\mathcal{T}$, and
- if a node $v = n_1 \dots n_j$ is in $\mathcal{T}$, then $v' = n_1 \dots n_{j-1} i$ is also in $\mathcal{T}$ for $0 < i < j$; we call v' an *older sibling* of v (and v a *younger sibling* of v'), and denote the set of older siblings of v by $\mathsf{os}(v)$.

Thus, ordered trees are non-empty and closed under predecessors and older siblings.

Generalised history tree. A generalised history tree $\mathcal{G}$ over Q for k accepting sets is a triple $\mathcal{G} = (\mathcal{T}, l, h)$ such that:

- $\mathcal{T}$ is an ordered tree,
- $l : \mathcal{T} \to 2^Q \setminus \{\emptyset\}$ is a labelling function such that
  - $l(v) \subsetneq l(\mathsf{pred}(v))$ holds for all $\varepsilon \neq v \in \mathcal{T}$,
  - the intersection of the labels of two siblings is disjoint ($\forall v, v' \in \mathcal{T}.\ v \neq v' \wedge \mathsf{pred}(v) = \mathsf{pred}(v') \Rightarrow l(v) \cap l(v') = \emptyset$), and
  - the union of the labels of all siblings is *strictly* contained in the label of their predecessor ($\forall v \in \mathcal{T}\ \exists q \in l(v)\ \forall v' \in \mathcal{T}.\ v = \mathsf{pred}(v') \Rightarrow q \notin l(v')$), and
- $h : \mathcal{T} \to [k]$ is a function that labels every node with a natural number from $[k]$. We call $F_{h(v)}$ the *active* accepting set of v.

For a generalised history tree $\mathcal{G} = (\mathcal{T}, l, h)$, $(\mathcal{T}, l)$ is the history tree introduced in [17]. Generalised history trees are enriched by the second labelling function, h, that is used to relate nodes with a particular accepting set.

3.1 Determinisation construction

Let $\mathcal{A} = (\Sigma, Q, I, T, \{F_i \mid i \in [k]\})$ be a generalised Büchi automaton with $|Q| = n$ states and k accepting sets. We will construct an equivalent deterministic Rabin automaton $\mathcal{D} = (\Sigma, D, d_0, \delta, \{(A_i, R_i) \mid i \in J\})$.

[17] separates the transition mechanism from the acceptance condition. We follow the same procedure and describe the transition mechanism below.

- D is the set of generalised history trees over Q.
- d_0 is the generalised history tree $(\{\varepsilon\}, l : \varepsilon \mapsto I, h : \varepsilon \mapsto 1)$.
- J is the set of nodes that occur in some ordered tree of size n.
- For every tree $d \in D$ and letter $\sigma \in \Sigma$, the transition $d' = \delta(d, \sigma)$ is the result of the following sequence of transformations:
  1. *Raw update of l.* We update l to the function l_1 by assigning, for all $v \in \mathcal{T}$, $l_1 : v \mapsto \{q \in Q \mid \exists q' \in l(v).\ (q', \sigma, q) \in T\}$, i.e., to the σ successors of $l(v)$.
  2. *Sprouting new children.* For every node $v \in d$ with c children, we sprout a new child vc. Let $\mathcal{T}_n$ be the tree of new children. Then we define, for all v in T_n, $l_1 : v \mapsto \{q \in Q \mid \exists q' \in l(\mathsf{pred}(v)).\ (q', \sigma, q) \in f_{h(\mathsf{pred}(v))}\}$, i.e., to the σ successors of the active accepting sets of their parents, and extend h to $T'_n = \mathcal{T} \cup \mathcal{T}_n$ by $h : v \mapsto 1$ for all $v \in \mathcal{T}_n$
  3. *Stealing of labels.* We obtain a function l_2 from l_1 by removing, for every node v with label $l(v) = Q'$ and all states $q \in Q'$, q from the labels of all younger siblings of v and all of their descendants.
  4. *Accepting and removing.* We denote with $\mathcal{T}_r \subseteq T'_n$ the set of all nodes v whose label $l_2(v)$ is now equal to the union of the labels of its children. We obtain $\mathcal{T}'$ from $\mathcal{T}'_n$ by removing all descendants of nodes in $\mathcal{T}_r$, and restrict the domain of l_2 and h accordingly. (The resulting tree $\mathcal{T}'$ may no longer be ordered.)
     Nodes in $\mathcal{T}' \cap \mathcal{T}_r$ are called *accepting*. We obtain h_1 from h by choosing $h : v \mapsto h(v) + 1$ for accepting nodes v with $h(k) \neq k$, $h_1 : v \mapsto 1$ for accepting nodes v with $h(k) = k$, and $h_1 : v \mapsto h(v)$ for all non-accepting nodes.
     The transition is in A_v iff v is accepting.
  5. *Renaming and rejecting.* To repair the orderedness, we call $\|v\| = |\mathsf{os}(v) \cap \mathcal{T}'|$ the number of (still existing) older siblings of v, and map $v = n_1 \dots n_j$ to $v' = \|n_1\|\ \|n_1 n_2\|\ \|n_1 n_2 n_3\| \dots \|v\|$, denoted $\mathsf{rename}(v)$.
     We update a triple $(\mathcal{T}', l_2, h_1)$ from the previous step to $d' = \big(\mathsf{rename}(\mathcal{T}'), l', h'\big)$ with $l' : \mathsf{rename}(v) \mapsto l_2(v)$ and $h' : \mathsf{rename}(v) \mapsto h_1(v)$.
     We call a node $v \in \mathcal{T}' \cap \mathcal{T}$ *stable* if $v = \mathsf{rename}(v)$, and we call all nodes in J *rejecting* if they are not stable. The transition is in R_v iff v is rejecting.

3.2 Correctness

In order to establish the correctness of our determinisation construction, we need to prove that $L(\mathcal{A}) = L(\mathcal{D})$, that is, we need to ascertain that the ω-language accepted by the nondeterministic generalised Büchi automaton is equivalent to the ω-language accepted by the deterministic Rabin automaton.

Theorem 1 **($L(\mathcal{D}) \subseteq L(\mathcal{A})$).** *Given that there is a node $v \in d$ (where d is a generalised history tree) which is eventually always stable and always eventually accepting for an ω-word α, then there is an accepting run of $\mathcal{A}$ on α.*

Notation. For an ω-word α and $j \geq i$, we denote with $\alpha[i,j[$ the word $\alpha(i)\alpha(i+1)\alpha(i+2)\ldots\alpha(j-1)$. We denote with $Q_1 \to^{\alpha} Q_2$ for a finite word $\alpha = \alpha_1\ldots\alpha_{j-1}$ that there is, for all $q_j \in Q_2$ a sequence $q_1\ldots q_j$ with $q_1 \in Q_1$ and $(q_i,\alpha_i,q_{i+1}) \in T$ for all $1 \leq i < j$. If, for all $q_j \in Q_2$, there is such a sequence that contains a transition in F_a, we write $Q_1 \Rightarrow_a^{\alpha} Q_2$.

Proof. Let $\alpha \in L(\mathcal{D})$. Then there is a v that is eventually always stable and always eventually accepting in the run $\rho_{\mathcal{D}}$ of $\mathcal{D}$ on α. We pick such a v.

Let $i_0 < i_1 < i_2 < \ldots$ be an infinite ascending chain of indices such that

- v is stable for all transitions $(d_j,\alpha(j),d_{j+1})$ with $j \geq i_0$, and
- the chain $i_0 < i_1 < i_2 < \ldots$ contains exactly those indices $i \geq i_0$ such that $(d_{i-1},\alpha(i-1),d_i)$ is accepting; this implies that h is updated exactly at these indices.

Let $d_i = (\mathcal{T}_i,l_i,h_i)$ for all $i \in \omega$. By construction, we have

- $I \to^{\alpha[0,i_0[} l_{i_0}(v)$, and
- $l_{i_j}(v) \Rightarrow_{h_{i_j}}^{\alpha[i_j,i_{j+1}[} l_{i_{j+1}}(v)$.

Exploiting König's lemma, this provides us with the existence of a run of $\mathcal{A}$ on α that visits all accepting sets F_i of $\mathcal{A}$ infinitely many times. (Note that the value of h is circulating in the successive sequences of the run.) This run is accepting, and α therefore in the language of $\mathcal{A}$. □

Theorem 2 **($L(\mathcal{A}) \subseteq L(\mathcal{D})$).** *Given that there is an accepting run of $\mathcal{A}$, there is a node which is eventually always stable and always eventually accepting.*

Notation. For a state q of $\mathcal{A}$ and a generalised history tree $d = (\mathcal{T},l,h)$, we call a node a *host node of q*, denoted $\mathsf{host}(q,d)$, if $q \in l(v)$ but not in $l(vc)$ for any child vc of v.

Proof. We fix an accepting run $\rho = q_0q_1\ldots$ of $\mathcal{A}$ on an input word α, and let $\rho_{\mathcal{D}} = d_0d_1\ldots$ be the run of $\mathcal{D}$ on α. We then define the related sequence of host nodes $\vartheta = v_0v_1v_2\ldots = \mathsf{host}(q_0,d_0)\mathsf{host}(q_1,d_1)\mathsf{host}(q_2,d_2)\ldots$. Let l be the shortest length $|v_i|$ of these nodes that occurs infinitely many times.

We follow the run and see that the initial sequence of length l of the nodes in ϑ eventually stabilises. Let $i_0 < i_1 < i_2 < \ldots$ be an infinite ascending chain of indices such that the length $|v_j| \geq l$ of the j-th node is not smaller than l for all $j \geq i_0$, and equal to $l = |v_i|$ for all indices $i \in \{i_0,i_1,i_2,\ldots\}$ in this chain. This implies that $v_{i_0},v_{i_1},v_{i_2},\ldots$ is a descending chain when the single nodes v_i are compared by lexicographic order. As the domain is finite, almost all elements of the descending chain are equal, say $v_i := \pi$. In particular, π is eventually always stable.

Let us assume for the sake of contradiction, that this stable prefix π is accepting only finitely many times. We choose an index i from the chain $i_0 < i_1 < i_2 < \ldots$ such that π is stable for all $j \geq i$. (Note that π is the host of q_i for d_i, and $q_j \in l_j(\pi)$ holds for all $j \geq i$.)

As ρ is accepting, there is a smallest index $j > i$ such that $(q_{j-1},\alpha(j-1),q_j) \in F_{h_i(\pi)}$. Now, as π is not accepting, q_i must henceforth be in the label of a child of π, which contradicts the assumption that infinitely many nodes in ϑ have length $|\pi|$.

Thus, π is eventually always stable and always eventually accepting. □

Corollary 1 ($L(\mathcal{A}) = L(\mathcal{D})$)**.** *The deterministic Rabin automaton generated by our determinisation procedure is language equivalent to the original generalised Büchi automaton.*

4 Complementation

In this section we connect determinisation and complementation. In order to construct a concise data structure for complementation, we first show that we can cut acceptance into two phases: a finite phase where we track only the reachable states, and an infinite phase where we also track acceptance. We then use this simple observation to devise an abstract complementation procedure, and then suggest a succinct data structure for it.

We first argue that acceptance of a word $\alpha \cdot \alpha'$ with $\alpha \in \Sigma^*$ and $\alpha' \in \Sigma^\omega$ depends only on α' and the states reachable through α.

Lemma 1. *If* $I \to^\alpha Q \Leftrightarrow I \to^\beta Q$ *then* $\alpha \cdot \alpha' \in L(\mathcal{A}) \Leftrightarrow \beta \cdot \alpha' \in L(\mathcal{A})$.

Proof. It is easy to see how an accepting run of $\mathcal{A}$ on $\alpha \cdot \alpha'$ can be turned into an accepting run on $\beta \cdot \alpha'$, and vice versa. □

This provides us with the following abstract description of a nondeterministic acceptance mechanism for the complement language of $\mathcal{A}$.

1. When reading an ω-word α, we first keep track of the reachable states R for a finite amount of time. (subset construction)
2. Eventually, we swap to a tree that consists only of nodes that are henceforth stable, and that are the only nodes that are henceforth stable, such that none of these nodes is henceforth accepting.
3. We verify the property described in (2).

Lemma 2. *The abstract decision procedure accepts an input word iff it is rejected by the deterministic Rabin automaton* $\mathcal{D}$*.*

Proof. The 'only if' direction follows directly from the previous lemma.

For the 'if' direction, we can guess a point i in the run of $\mathcal{D}$ on α where all eventually stable nodes are introduced and stable, and none of them is henceforth accepting. We claim that we can simply guess this point of time, but instead of going to the respective generalised tree $d_i = (\mathcal{T}, l, h)$, we go to $d'_i = (\mathcal{T}', l', h')$, where $\mathcal{T}'$ is the restriction of $\mathcal{T}$ to the henceforth stable states, and l' and h' are the restrictions of l and h to $\mathcal{T}'$. (Note that the subtree of henceforth stable nodes is always ordered.)

Clearly, all nodes in $\mathcal{T}'$ are stable, and none of them are accepting in the future. It remains to show that none of their descendants is stable. Assume one of the children a node $v \in \mathcal{T}'$ spawns eventually is stable. We now consider a part of the 'run' of our mechanism starting at i, $d'_i d'_{i+1} d'_{i+2} \ldots$. Invoking König's Lemma, we get a run $\rho = q_0 q_1 \ldots$ such that, for some $j > i$ and for all $m > j$, some $v_j = \mathsf{host}(q_j, d'_j)$, which is a true descendant of v, is the host of q_j. Using a simple inductive argument that exploits that v is henceforth stable but not accepting, this implies for the run $\rho = d_0 d_1 \ldots$ that, for the same $j > i$ and for all $m > j$, some $v'_j = \mathsf{host}(q_j, d'_j)$, which is a true descendant of v, is a host of q_j. This implies in turn that some descendant of v is eventually stable and thus leads to a contradiction. □

We call an ordered tree *flat* if it contains only nodes of length ≤ 1.

Lemma 3. *We can restrict the choice in (2) to flat trees.*

Proof. If we rearrange the nodes in $\mathcal{T}$ following the "stealing and hosting order", that is, mapping a node v with length ≥ 1 to a smaller node v' with length ≥ 1 if either v' is an ancestor of v or an initial sequence of v is an older sibling of an initial sequence of v'. This describes a unique bijection $b : \mathcal{T} \to \mathcal{F}$, where $\mathcal{F}$ is the flat tree with $|\mathcal{T}| = |\mathcal{F}|$, and we choose $d'_i = (\mathcal{F}, l' : b(v) \mapsto l(\mathsf{pred}(b(v))) \cup l(v) \setminus l(\mathsf{pred}(v)), h' : b(v) \mapsto h(v))$ instead of $d_i = (\mathcal{T}, l, h)$. (The complicated looking $l' : b(v) \mapsto l(\mathsf{pred}(b(v))) \cup l(v) \setminus l(\mathsf{pred}(v))$ just means $v = \mathsf{host}(q, d_i) \Leftrightarrow b(v) = \mathsf{host}(q, d'_i)$, that is, the hosts are moved, not the full label.)

It is easy to see that, if we compare two runs starting in d_i and d'_i on any word, they keep this relation. □

To obtain a succinct data structure for the second phase of the run, we do not follow the precise development of the individual new children of the henceforth stable nodes, but rather follow simple subset constructions. One subset that is kept for all stable nodes is the *union* of the nodes of its children. Note that, to keep track of this union, it is not necessary to keep track of the distribution of these sets to the different children (let alone their descendants).

To check that all children spawned at a particular point j in a run will eventually be deleted, one can keep track of an additional subset: the union of all labels of nodes of children that already existed in j. If this subset runs empty, then all of these children have been removed. Vice versa, if all of these children are removed, then this subset runs empty.

Note that these subsets are, in contrast to the nodes in the flat generalised history tree, not numbered. For efficiency, note that it suffices to use the second subset for only one of the nodes in the flat trees at a time, changing this node in a round robin fashion.

Theorem 3. *The algorithm outlined above describes a nondeterministic Büchi automaton that recognises the complement of the language of $\mathcal{A}$.*

The trees and sets we need can be encoded using the following data structure. We first enrich the set of states by a fresh marker m, used to mark the extra subset for new children in the stable node under consideration, to $Q_m = Q \cup \{m\}$. We then add the normal subsets that capture the label of all children as a child of each stable state as a single child of this state. For a single stable state that we currently track, we add a (possibly second and then younger) child for new children. The labelling is as described above, except that m is added to the label for new children (which otherwise might be empty) and its ancestors.

We can now choose $h : v \to k + 1$ for all non-stable nodes v in this tree. As this naming convention clearly identifies these nodes, we can represent the tree as a flat tree.

4.1 Complexity of complementing generalised Büchi automata

In this section, we establish lower bounds for the complementation of generalised Büchi automata and show that the construction we outlined tightly meets these lower bounds. Our lower bound proof builds on *full* automata. A generalised Büchi automaton $\mathcal{B}_n^k = (\Sigma, Q, I, T, \{F_i \mid i \in [k]\})$ is called *full* if

- $\Sigma_n^k = 2^{Q \times [k+1] \times Q}$, $|Q| = n$, and $I = Q$,
- $T = \{(q, \sigma, q') \mid \exists i \in [k+1].\ (q, i, q') \in \sigma\}$, and
- $F_i = \{(q, \sigma, q') \mid (q, i, q') \in \sigma\}$.

As each generalised Büchi automaton with n states and k accepting sets can be viewed as a language restriction (by alphabet projection) of a full automaton, full automata are useful tools in establishing lower bounds. We show that, for each $\mathcal{B}_n^k$, there is a family of $L_n^k \subseteq \Sigma_n^k$ such that a^ω is not in the language of $\mathcal{B}_n^k$ for any $a \in L_n^k$, and each nondeterministic generalised Büchi automaton that recognises the complement language of $\mathcal{B}_n^k$ must have at least $|L_n^k|$ states. The size of this alphabet is such that the size of $\mathcal{B}_n^k$ is between $|L_n^k|$ and $|L_{n+1}^{k+1}|$, which provides us with tight bounds for the complementation of generalised Büchi automata.

Let us first define the letters in L_n^k. We call a function $f : Q \to \mathbb{N}$ *full* if its domain is $[n]$ for some n. Let f be a full function with domain $[n]$ then we call a function $f_\# : [n] \to [k]$ a *k-numbering* of f. We denote by $\mathsf{enc}(f, f_\#)$ the letter encoding a function f with k-numbering $f_\#$ as the letter that satisfies

- $(p, \mathsf{enc}(f, f_\#), q) \in T$ iff $f(q) \leq f(p)$, and
- $(p, \mathsf{enc}(f, f_\#), q) \in F_b$ iff *either* $f(q) < f(p)$ *or* $(f(p) = f(q)$ and $f_\#\big(f(p)\big) \neq b)$.

Obviously, if two full functions f, g with respective k-numberings $f_\#, g_\#$ encode the same letter $\mathsf{enc}(f, f_\#) = \mathsf{enc}(g, g_\#)$, then they are equal. First, we note that the word a^ω is not in the language of $\mathcal{B}_n^k$.

Lemma 4. *Let f be a full function, $f_\#$ be a k-numbering of f, and let a be the letter encoded by f and $f_\#$. Then a^ω is rejected by $\mathcal{B}_n^k$.*

Proof. Assume, for contradiction, that there is an accepting run ρ of $\mathcal{B}_n^k$ on a^ω. By the definition of an encoding, the sequence $f_i = f(\rho_i)$ is monotonously decreasing. It will therefore stabilise eventually, say at j. (I.e., $\forall l \geq j.\ f(\rho_l) = f(\rho_j)$.) By the definition of accepting transitions for encoded letters there will henceforth be no more transition from the final transitions $F_{f_\#(f_j)}$. $\lightning$ □

We now define $L_n^k = \{\mathsf{enc}(f, f_\#) \mid f \text{ is full and } f_\# \text{ is a } k\text{-numbering of } f\}$.

Theorem 4. *A generalised Büchi automaton $\mathcal{C}_n^k$ that recognises the complement language of $\mathcal{B}_n^k$ has at least $|L_n^k|$ states.*

Proof. The previous lemma establishes that a^ω is accepted by $\mathcal{C}_n^k$. We choose accepting runs ρ_a with infinity set I_a for each letter $a \in L_n^k$, and show by contradiction that I_a and I_b are disjoint for two different letters $a, b \in L_n^k$.

Assume that this is not the case for two different letters a and b. It is then simple to infer from their accepting runs ρ_a and ρ_b natural numbers $l,m,n,a \in \mathbb{N}$ such that $\rho = \rho_a[0,l](\rho_b[a,a+m]\rho_a[l,l+n])^\omega$ is accepting. Then $w = a^l(b^m a^n)^\omega$ is accepted by $\mathcal{C}_n^k$, as ρ is a run of w. We lead this to a contradiction by showing that w is in the language of $\mathcal{B}_n^k$.

We have $a = \mathsf{enc}(f, f_\#) \neq b = \mathsf{enc}(g, g_\#)$. Let us first assume $f = g$. Then $f_\# \neq g_\#$, and we can first choose an i with $f_\#(i) \neq g_\#(i)$ and then a q with $f(q) = i$. It is now simple to construct an accepting run with trace q^ω for $\mathcal{B}_n^k$ for w. ↯

Let us now assume $f \neq g$. We then set i to the minimal number such that f and g differ in i ($f^{-1}(i) \neq g^{-1}(d)$, where $^{-1}$ denotes the preimage of i. W.l.o.g., we assume $f^{-1}(i) \setminus g^{-1}(i) \neq \emptyset$. We choose a $q \in f^{-1}(i) \setminus g^{-1}(i)$. It is now again simple to construct an accepting run with trace q^ω for $\mathcal{B}_n^k$ for w. ↯

This closes the case distinction and provides the main contradiction. □

The only thing that remains to be shown is tightness. But there is obviously an injection from the flat trees described at the end of the complementation (plus the subsets) of an automaton with n states and k accepting pairs into L_{n+1}^{k+1}. This provides:

Theorem 5. $|L_{n+1}^{k+1}| > |\mathcal{B}_n^k|$.

This provides bounds which are tight in k and n with a negligible margin of 1. For large k, the size $|L_n^k|$ can be approximated by $\left(\frac{kn}{e}\right)^n$: It is not hard to show that the size of L_n^k is dominated by encodings that refer to functions from $[n]$ onto $[n]$, and the number of these encodings is $n!k^n$. (E.g., $|L_n^n| < (e-1)n!n^n$.)

Our conjecture is that the construction is tight at least in n. The reason for this assumption is that the increment in n stems from the round robin construction that keeps track of the stable node under consideration, while the alphabet L_n^k refers to the far more restricted case that stable states never spawn new children, rather than merely requiring that none of the children spawned is henceforth stable.

5 Optimality of our determinisation construction

Colcombet and Zdanowski have shown that the determinisation technique we use (and used in [17]) is optimal for the case of determinising Büchi automata. Their proof is by reducing the resulting deterministic Rabin automaton to a game and citing the memory required for the winner of the game to establish a lower bound on the size of the deterministic Rabin automaton [2]. In this section, we extend their result to the case of generalised Büchi automata. We show that the function $\mathsf{ghist}_k(n)$, that maps n to the number of generalised history trees for k accepting sets—and hence to the number of states of the resulting deterministic Rabin automaton obtained by our determinisation construction—is also a lower bound for the number of states needed for language equivalent deterministic Rabin automata.

5.1 Games

We follow the conventions defined in [2]. A *two player* game is a tuple $G = (V, E, L, W)$, where V is a set of states of the game, which is partitioned into V_0 and V_1, states for the

two players, Player 0 and Player 1 respectively, $E \in V \times L \times V$ is the transition relation and $W \in L^{\omega}$ is the winning condition. We require that, for every $v \in V$, there exists some state $v' \in V$ such that $(v,a,v') \in E$ is a transition from the location v to v'. We say that (v,a,v') produces the letter a.

A *play* of the game G is a maximal sequence of locations $p = (v_0, a_0, v_1, a_1 \ldots)$ such that, for all $i \geq 0$, we have $(v_i, a_i, v_{i+1}) \in E$. Let $p_L = (a_0 a_1 \ldots)$ be the sequence of letters generated by the play p. Player 0 wins the play p if $p_L \in W$. Player 1 wins otherwise.

A strategy for player X is a function that specifies how the player should play depending on the history of transitions. A *strategy* for player X is a function σ mapping finite sequences $(v_0, a_0, \ldots, a_{n-1}, v_n)$ into E. A play p is *compatible* with a strategy σ for player 0 if, for all prefixes $(v_0, a_0, \ldots, v_{n-1}, a_{n-1}, v_n)$ of p, $\sigma(v_0, \ldots v_{n-1}) = (v_{n-1}, a_{n-1}, v_n)$ if $v_{n-1} \in V_0$. A strategy σ for player 0 is a *winning strategy* if player 0 wins every play compatible with σ. Strategies for player 1 are defined similarly, with V_1 instead of V_0.

A strategy with *memory of size m* for player 0 is a tuple $(M, update, choice, init)$, where M is a set of size m, called the memory, $update$ is a mapping from $M \times E$ to M, *choice* is a mapping from $V_0 \times M$ to E, and $init \in M$. Player X wins a game with memory of size m if she has a winning strategy with memory of size m. When we have a strategy with memory of size 1, we call it a *positional strategy*.

A game G is called a *Rabin game* if the winning condition W is described by a Rabin condition over the transitions of this game. For every *Rabin game*, player 0 can win using a positional strategy [21].

Finite games of infinite duration can effectively be reduced to deterministic automata and vice-versa. Given a deterministic Rabin automaton $\mathcal{D}$ with n states that accepts the language W of a generalised Büchi automaton $\mathcal{A}$, and a game with winning condition W, one can construct the product of the automaton $\mathcal{D}$ with the game, and derive a game with the Rabin condition as the winning condition of the game. It is obvious that such a game (with its Rabin condition) admits positional strategies and that the deterministic Rabin automaton maintains the memory for a strategy in the original game with the generalised Büchi winning condition.

Lemma 5. *[2] If player 0 wins a game with the winning condition W (the generalised Büchi condition) while requiring memory of size n, then every deterministic Rabin automaton that is language equivalent to W has at least n states.*

5.2 Proving a lower bound

Lemma 5 provides a viable argument to prove a lower bound on the determinisation of generalised Büchi automata. For this, we use the full automata introduced in the previous section.

Closely following Colcombet & Zdanowski's proof [2], we use $\mathcal{B}_n^k = (\Sigma, Q, I, T, \{F_i \mid i \in [k]\})$. To prove our lower bound, we first restrict ourselves to a constant set of reachable states. We then proceed by proving a (tight) bound within this restricted scenario. Finally, we extend this lower bound to the general case.

Restricting the set of reachable states. We define the set of states reachable by a word u, Reach(u) by induction. Obviously, Reach(ε) $= Q$. For $v \in \Sigma^*$ and $a \in \Sigma$, we define Reach(va) as follows: for all $q \in Q$ and $q' \in Q$, $q' \in$ Reach(va) iff $q \in$ Reach(v) and there is a transition $(q,i,q') \in a$. Let Σ_S be the set of letters $a \in \Sigma$ such that Reach(a) $= S$ and Reach(v) $= S$ implies Reach(va) $= S$. Let $\mathcal{L}(\mathcal{B}_S^k)$ be $\mathcal{L}(\mathcal{B}_n^k) \cap {\Sigma_S}^\omega$.

Each generalised history tree $d \in D$ maintains the set of states reachable by the generalised Büchi automaton at the current position of the input word in the label $l(\varepsilon)$ of its root. Thus, if we restrict ourselves to a set of states $S \subseteq Q$, it is enough if we consider the set of generalised history trees D_S, which contain the states $S \subseteq Q$ in the labels of their root ε. The runs of the deterministic Rabin automaton $\mathcal{D}_n^k$, which we get when determinising $\mathcal{B}_n^k$, on an ω-word $\alpha \in {\Sigma_S}^\omega$, is therefore a sequence in ${D_S}^\omega$.

A game to prove tightness. In this restricted context, we define a game G such that player 0 wins G, requiring at least memory $|D_S|$, which will in turn establish that any deterministic Rabin automaton accepting $\mathcal{L}(\mathcal{B}_S^k)$ has at least $|D_S|$ states.

We define the game G with the winning condition $\mathcal{L}(\mathcal{B}_S^k)$. Formally, $G = (V,E,L,W)$ where V is the set of states of the game partitioned into V_0 and V_1, the states for players 0 and 1 respectively. V_0 is a singleton set $\{v_0\}$ and V_1 consists of the initial state of the game v_{in} and one position v_d for each generalised history tree $d \in D_S$. The labelling function L is the alphabet Σ_S^+. The winning condition W is $\mathcal{L}(\mathcal{B}_S^k)$. The game transitions E are as follows:

- $(v_{in}, u, v_0) \in E$ for all $u \in \Sigma_S^+$,
- $(v_0, id_S, v_d) \in E$ for each generalised history tree d, where $id_S = \{(q, k+1, q) \mid q \in S\}$ is the input letter that maintains all trees in D_S,
- $(v_d, u, v_0) \in E$ for each generalised history tree $d \in D_S$ and word $u \in \Sigma_S^+$ if u is *profitable* for d. We call u profitable if there is a v in the ordered tree of d such that
  - v is stable throughout the sequence of a run of $\mathcal{D}_n^k$ starting in d when u is read, and
  - v is accepting for some transition in the sequence.

Player 0 has a simple winning strategy in this game. It suffices if player 0 keeps track of the current state of $\mathcal{D}_n^k$ when following the ω-word produced in this game. When it is her turn and the run is in state d, then she plays to v_d. This way, player 1 is forced to play only profitable words, which leads to a minimum node that is always eventually profitable, and hence to acceptance [2].

Lemma 6. *Player 0 has a winning strategy in the game G.*

Modified game to prove memory lower bound. We define a modified game G_{mod} by removing one of player 1's game states v_d, thereby denying player 0 the corresponding move. This is the only difference to the game G. This lemma is the technical core of the proof requiring an analysis of the differences between two trees.

Lemma 7. *Let $d \neq d'$ be generalised history trees in D_S. There exists a word u such that*

– $(v_{d'}, u, v_0)$ *is a move in* G_{mod},
– $d \to^u d$,
– u *is not profitable for* d.

Proof. We distinguish two cases. First, we assume that $d = (\mathcal{T}, l, h)$ and $d' = (\mathcal{T}, l, h')$. This is the easy part: we can simply use a node $v \in \mathcal{T}$ such that $h(v) \neq h'(v)$, but this does not hold for any descendant of v. We then choose $Q' = \{q \in S \mid v = \mathsf{host}(q, d)\}$ to be the set of nodes hosted by v. (Note that these are the same for d and d'.)

In this case, we can simply play the one letter word $\alpha = id_S \cup \{(q, h'(v), q) \mid q \in Q'\}$, which satisfies all the properties from above: clearly the transition is profitable for $v_{d'}$ (as v is accepting in the respective transition $d' \to^\alpha d''$) whereas $d \to^\alpha d$ holds while none of the nodes of $\mathcal{T}$ is accepting.

Now we assume that $d = (\mathcal{T}, l, h)$ and $d' = (\mathcal{T}', l', h')$ with $(\mathcal{T}', l') \neq (\mathcal{T}, l)$. But for this case, we can almost use the same strategy for choosing a finite word u as for ordinary history trees [2]. The extra challenge is that, when reconstructing d, it is not enough to spawn a new child, we also have to update h, which can be done using a sequence of letters like the letter α from above after reconstructing a node v. □

With the help of this lemma, we can prove the following.

Lemma 8. *For every* $d \in D_S$*, player 1 has a winning strategy in* G_{mod}.

A winning strategy for player 1 is as follows. From v_{in}, player 1 plays a word u such that $d_0 \to^u d$, where d_0 is the initial state of the deterministic Rabin automaton. A good response from player 0 would be to move to v_d. But v_d has been removed and player 0 has to move to a different game state, say $v_{d'}$ for some $d' \neq d$. Player 1 responds according to Lemma 7. Using this strategy, player 1 ensures that, when the play gets infinite, there is no node that is always eventually accepting and the deterministic Rabin automaton $\mathcal{D}$ does not accept this word. Hence, $u \notin \mathcal{L}(\mathcal{B}_S^k)$ and player 1 wins.

Corollary 2. *Player 0 has no winning strategy with memory* $|D_S| - 1$ *in the game G.*

If player 0 had a winning strategy with memory $|D_S| - 1$, then there would be a game state v_d which is never visited by this strategy. But this would also mean that player 0 can win G_{mod} with this strategy, which contradicts Lemma 8. Using Lemma 5, we now obtain:

Theorem 6. *Every deterministic Rabin automaton that accepts* $\mathcal{L}(\mathcal{B}_S^k)$ *has size at least* $|D_S|$.

Extending the lower bound to the unrestricted case. We have proven a lower bound for the case where we restricted the set of reachable states. We now extend our result to the general case. We do this by decomposing the deterministic Rabin automaton accepting $L(\mathcal{B}_n^k)$ into disjoint sets of states. Such a decomposition is feasible due to the following lemma.

Lemma 9. *[2] Let* $\mathcal{D}$ *be the deterministic Rabin automaton accepting* $\mathcal{L}(\mathcal{B}_n^k)$ *with transition function* δ *and initial state* d_0*. If* $\delta(d_0, u) = \delta(d_0, v)$*, then Reach*$(u)$ =*Reach*(v).

The above lemma describes the scenario where we restricted the set of reachable states, considering only the set of generalised history trees D_S. The automaton $\mathcal{D}$ restricted to D_S can be seen as a Rabin automaton which accepts $\mathcal{L}(\mathcal{B}_n^k)$ with restriction to letters in Σ_S. Because the sets D_S are disjoint, we have

$$\mathsf{ghist}_k(n) = |D| = \sum_{S \subseteq Q} |D_S|.$$

Theorem 7. *Every deterministic Rabin automaton accepting $\mathcal{L}(\mathcal{B}_n^k)$ has size at least* $\mathsf{ghist}_k(n)$.

ghist_k can be estimated in a similar way as the number of history trees for the determinisation of Büchi automata [17], as generalised history trees are, just like history trees, ordered trees with further functions on the set of nodes of the tree. Using the functions from [17] $\mathsf{ghist}_k(n) \in \sup_{x>0} O\big(m(x) \cdot k^{\beta(x)} \cdot 4^{\beta(x)}\big)$, providing

$(1.65\,n)^n$, for $k = 1$, $(3.13\,n)^n$ for $k = 2$, and $(4.62n)^n$ for $k = 3$.

This value converges against

$$(1.47kn)^n$$

for large k.

Note that, when the generalised Büchi automaton we start with has exactly one accepting set, our bound construction coincides with [17].

References

1. J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Methodology, and Philosophy of Science, 1960, Berkeley, California, USA*, pages 1–11. Stanford University Press, 1962.
2. T. Colcombet and K. Zdanowski. A tight lower bound for determinization of Büchi automata. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming, Part II (ICALP 2009), 5–12 July, Rhodes, Greece*, volume 5556 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.
3. R. Gerth, D. Dolech, D. Peled, M. Y. Vardi, P. Wolper, and U. D. Liege. Simple on-the-fly automatic verification of linear temporal logic. In *In Protocol Specification Testing and Verification*, pages 3–18. Chapman & Hall, 1995.
4. D. Kähler and T. Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part I (ICALP 2008), 6–13 July, Reykjavik, Iceland*, volume 5125 of *Lecture Notes in Computer Science*, pages 724–735. Springer-Verlag, 2008.
5. O. Kupferman, N. Piterman, and M. Vardi. Safraless compositional synthesis. In *Proceedings of Computer Aided Verification (CAV 2006)*, Lecture Notes in Computer Science, pages 31–44. Springer-Verlag, 2006.
6. O. Kupferman and M. Vardi. Safraless decision procedures. In *Proceedings 46th IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23–25 October, Pittsburgh, PA, USA*, pages 531–540, 2005.
7. W. Liu and J. Wang. A tighter analysis of Piterman's Büchi determinization. *Information Processing Letters*, 109:941–945, 2009.

8. R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, October 1966.
9. D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1-2):69–107, 1995.
10. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Journal of Logical Methods in Computer Science*, 3(3:5), 2007.
11. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977), 31 October–1 November, Providence, Rhode Island, USA*, pages 46–57. IEEE Computer Society Press, 1977.
12. M. O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the American Mathematical Society*, 141:1–35, 1969.
13. M. O. Rabin and D. S. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.
14. S. Safra. On the complexity of ω-automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS 1988), 24–26 October, White Plains, New York, USA*, pages 319–327. IEEE Computer Society Press, 1988.
15. S. Safra. Exponential determinization for omega-automata with strong-fairness acceptance condition (extended abstract). In *STOC*, pages 275–282, 1992.
16. S. Schewe. Büchi complementation made tight. In S. Albers and J.-Y. Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, volume 3 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 661–672, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
17. S. Schewe. Tighter bounds for the determinisation of Büchi automata. In *Proceedings of the Twelfth International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2009), 22–29 March, York, England, UK*, volume 5504 of *Lecture Notes in Computer Science*, pages 167–181. Springer-Verlag, 2009.
18. S. Schewe and B. Finkbeiner. Bounded synthesis. In *Proceedings of the 5th International Symposium on Automated Technology for Verification and Analysis (ATVA 2007), 22–25 October, Tokyo, Japan*, volume 4762 of *Lecture Notes in Computer Science*, pages 474–488. Springer-Verlag, 2007.
19. R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1/2):121–141, 1982.
20. Q. Yan. Lower bounds for complementation of *omega*-automata via the full automata technique. *Journal of Logical Methods in Computer Science*, 4(1:5), 2008.
21. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.