Complementing Semi-Deterministic Büchi Automata*

František Blahoudek[1], Matthias Heizmann[2], Sven Schewe[3], Jan Strejček[1], and Ming-Hsien Tsai[4]

[1] Masaryk University, Brno, Czech Republic
[2] University of Freiburg, Germany
[3] University of Liverpool, UK
[4] Institute of Information Science, Academia Sinica, Taiwan

Abstract. We introduce an efficient complementation technique for semi-deterministic Büchi automata, which are Büchi automata that are deterministic in the limit: from every accepting state onward, their behaviour is deterministic. It is interesting to study semi-deterministic automata, because they play a role in practical applications of automata theory, such as the analysis of Markov decision processes. Our motivation to study their complementation comes from the termination analysis implemented in ULTIMATE BÜCHI AUTOMIZER, where these automata represent checked runs and have to be complemented to identify runs to be checked. We show that semi-determinism leads to a simpler complementation procedure: an extended breakpoint construction that allows for symbolic implementation. It also leads to significantly improved bounds as the complement of a semi-deterministic automaton with n states has less than 4^n states. Moreover, the resulting automaton is unambiguous, which again offers new applications, like the analysis of Markov chains. We have evaluated our construction against the semi-deterministic automata produced by the ULTIMATE BÜCHI AUTOMIZER. The evaluation confirms that our algorithm outperforms the known complementation techniques for general nondeterministic Büchi automata.

1 Introduction

The complementation of Büchi automata [6] is a classic problem that has been extensively studied [6,22,19,26,23,31,20,25,17,13,12,37,27,33,11,32] for more than half a century; see [35] for a survey. The traditional line of research has started with a proof on the existence of complementation algorithms [22,19] and continued to home in on the complexity of Büchi complementation, finally leading to matching upper [27] and lower [37] bounds for complementing Büchi automata.

* The research was supported through The Czech Science Foundation, grant P202/12/G061, by the German Research Council (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR14 AVACS) and EPSRC grant EP/M027287/1.

This line of research has been extended to more general classes of automata, notably parity [30] and generalised Büchi [29] automata.

The complementation of Büchi automata is a valuable tool in formal verification (cf. [18]), in particular when a property that all runs of a model shall have is provided as a Büchi automaton,[1] and when studying language inclusion problems of ω-regular languages. With the growing understanding of the worst case complexity, the practical cost of complementing Büchi automata has become a second line of research. In particular the GOAL tool suite [33] provides a platform for comparing the behaviour of different complementation techniques on various benchmarks [32].

While these benchmarks use general Büchi automata, practical applications can produce or require subclasses of Büchi automata in specific forms. Our research is motivated by the observation that the program termination analysis in Ultimate Büchi Automizer [15] and the LTL software model checker Ultimate LTL Automizer [9] produce *semi-deterministic Büchi automata* (SDBA) [34,36] during their run. Semi-deterministic Büchi automata are a special class of Büchi automata that behave deterministically after traversing the first accepting state. For this reason, they are sometimes referred to as *limit deterministic* or *deterministic-in-the-limit* Büchi automata.

Program termination analysis is a model checking problem, where the aim is to prove that a given program terminates on all inputs. In other words, it tries to establish (or disprove) that all infinite execution paths in the program flowgraph are infeasible. The Ultimate Büchi Automizer uses an SDBA to represent infinite paths that are already known to be infeasible. It needs to complement the SDBA and make the product with the program flowgraph to identify the set of infinite execution paths whose infeasibility still needs to be proven. One can use off-the-shelf complementation algorithms like rank based [17,13,12,27] or determinisation based [25,24,28,29] ones, but they make no use of the special structure of SDBAs.

We show that exploiting this structure helps: while the complementation of Büchi automata with n states leads to a $(cn)^n$ blow-up for a constant $c \approx 0.76$ (cf. [27] for the upper and [37] for the lower bound), an SDBA with n states can be complemented to an automaton with less than 4^n states. More precisely, if the deterministic part (the states reachable from the accepting states) contains d states, including a accepting states, the complement automaton has at most $2^{n-d}3^a4^{d-a}$ states. The $2^{\Theta(n)}$ blow-up is tight as an $\Omega(2^n)$ lower bound is inherited from the complementation of nondeterministic finite automata. Another advantage of our construction is that it is suitable for the simplest class of Büchi automata: deterministic Büchi automata with a accepting and n non-accepting states are translated to $2n - a$ states, which meets Kurshan's construction for the complementation of deterministic Büchi automata [18].

[1] In model checking, one tests for emptiness the intersection of the automaton that recognises the runs of a system with the automaton that recognises the complement of the property language.

Moreover, the resulting automata have further useful properties. For example, their structure is very simple: they are merely an extended breakpoint construction [21]. Like ordinary breakpoint constructions, this provides a structure that is well suited for symbolic implementation. This is quite different from techniques based on Safra style determinisation [25,24,28,29]. In addition to this, they are *unambiguous*, i.e. there is exactly one accepting run for each word accepted by such an automaton. This is notable, because disambiguation is another automata transformation that seems to be more involved than complementation, but simpler than determinisation [16], and it has proven to be useful for the quantitative analysis of Markov chains [7,3]. For our motivating application, this is particular good news, as the connection to Markov chains implies direct applicability to model checking stochastic models as well as nondeterministic ones. The connection to stochastic models closes a cycle of applications, as they form a second source for applying semi-deterministic automata: they appear in the classic algorithm for the qualitative analysis of Markov decision processes [8] and in current model checking tools for their quantitative analysis [14] alike.

With all of these favourable properties in mind, it would be easy to think that the complementation mechanism we develop forms a class of its own. But this is not the case: when comparing it with classic rank based complementation [17] and its improvements [13,12,27], semi-deterministic automata prove to be automata, where all states in all runs can be assigned just three ranks, ranks 1 through 3 in the terminology of [17]. Consequently, there are only states with a single even rank, and a rank based algorithm that has to guess the rank correctly for states that are reachable from an accepting state has very similar properties. From this perspective, one could say that complementation and disambiguation are easy to obtain, as very little needs to be guessed (only the point where the rank of a state goes down to 1) and very little has to be checked.

We also motivate and present an on-the-fly modification of our complementation, which does not need to know the whole automaton before the complementation starts. The price for the on-the-fly approach is a slightly worse upper bound on the size of the produced automaton for the complement: it has less than 5^n states.

We have implemented our construction in the GOAL tool and the Ultimate Automata Library and evaluated it on semi-deterministic Büchi automata that were produced by Ultimate Büchi Automizer applied to programs of the Termination category of the software verification competition SV-COMP 2015 [4]. The evaluation confirms that the specific complementation algorithm realises its theoretical advantage and outperforms the traditional algorithms and produces smaller complement automata.

The remainder of the paper is organised as follows. After recalling some definitions and introducing our notation in Section 2, we present the complementation construction in Section 3 together with its complexity analysis and on-the-fly modification. In Section 4, we show a connection between our construction and rank-based constructions, followed by a correctness proof for our construction. The experimental evaluation is presented in Section 5.

2 Preliminaries

A *(nondeterministic) Büchi automaton* (NBA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, where

- Q is a finite set of *states*,
- Σ is a finite *alphabet*,
- $\delta : Q \times \Sigma \to 2^Q$ is a *transition function*,
- $I \subseteq Q$ is a set of *initial states*, and
- $F \subseteq Q$ is a set of *accepting states*.

A *run* of an automaton $\mathcal{A}$ over an infinite word $w = w_0 w_1 \ldots \in \Sigma^\omega$ is a finite or infinite sequence of states $\rho = q_0 q_1 q_2 \ldots \in Q^+ \cup Q^\omega$ such that $q_0 \in I$ and $q_{j+1} \in \delta(q_j, w_j)$ for each pair of adjacent states $q_j q_{j+1}$ in ρ. For a finite run $\rho = q_0 q_1 q_2 \ldots q_n \in Q^{n+1}$ we require that there is no transition for its last state, i.e. $\delta(q_n, w_n) = \emptyset$, and we say that the run *blocks*. A run is *accepting* if $q_j \in F$ holds for infinitely many j. A word w is *accepted* by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ over w. The *language* of an automaton $\mathcal{A}$ is the set $L(\mathcal{A})$ of all words accepted by $\mathcal{A}$.

A *complement* of a Büchi automaton $\mathcal{A}$ is a Büchi automaton $\mathcal{C}$ over the same alphabet Σ that accepts the complement language, $L(\mathcal{C}) = \Sigma^\omega \smallsetminus L(\mathcal{A})$, of the language of $\mathcal{A}$.

A Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ is called *complete* if, for each state $q \in Q$ and for each letter $a \in \Sigma$, there exists at least one successor, i.e. $|\delta(q, a)| \geq 1$. A Büchi automaton $\mathcal{A}$ is *unambiguous* if, for each $w \in L(\mathcal{A})$, there exists only one accepting run over w.

A state of a Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ is called *reachable* if it occurs in some run for some word $w \in \Sigma^\omega$. $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ is called *deterministic* if it has only one initial state, i.e. if $|I| = 1$, and if, for each reachable state $q \in Q$ and for each letter $a \in \Sigma$, there exists at most one successor, i.e. $|\delta(q, a)| \leq 1$.

We are particularly interested in semi-deterministic automata. A Büchi automaton is semi-deterministic if it behaves deterministically from the first visit of an accepting state onward. Formally, a Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ is a *semi-deterministic Büchi automaton* (SDBA) (also known as *deterministic-in-the-limit*) if, for each $q_f \in F$, the automaton $(Q, \Sigma, \delta, \{q_f\}, F)$ is deterministic.

Each semi-deterministic automaton can be divided into two parts: the part reachable from accepting states—which is completely deterministic—and the rest. Hence, one can alternatively define a semi-deterministic automaton such that the set of states $Q = Q_1 \cup Q_2$ consists of two disjoint sets Q_1 and Q_2, where $F \subseteq Q_2$, and the transition relation $\delta = \delta_1 \cup \delta_t \cup \delta_2$ consists of three disjoint transition functions, namely

$$\delta_1 : Q_1 \times \Sigma \to 2^{Q_1}, \qquad \delta_t : Q_1 \times \Sigma \to 2^{Q_2}, \text{ and } \qquad \delta_2 : Q_2 \times \Sigma \to 2^{Q_2},$$

where the relation δ_2 is deterministic: for each $q \in Q_2$ and each $a \in \Sigma$, $|\delta_2(q, a)| \leq 1$. δ can then be defined as $\delta(q, a) = \delta_1(q, a) \cup \delta_t(q, a)$ if $q \in Q_1$ and $\delta(q, a) = \delta_2(q, a)$ if $q \in Q_2$. The elements of δ_t are called *transit edges*. This alternative definition is captured in Figure 1 and used in the following section.

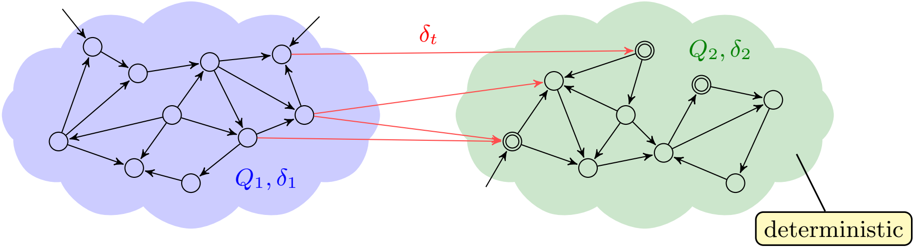

Fig. 1. A semi-deterministic Büchi automaton: δ_2 is deterministic, accepting states are only in Q_2, and transit edges (δ_t) lead from Q_1 to Q_2.

3 Semi-deterministic Büchi Automata Complementation

First of all, we explain our complementation construction intuitively. Then we formulate it precisely and discuss the size of the resulting automata when the complementation is applied to semi-deterministic and deterministic Büchi automata. At the end, we briefly introduce the modification of our complementation construction for on-the-fly approach. The correctness is addressed in Section 4 after introducing the concept of level rankings and run graphs.

3.1 Relation of Runs to the Complement

Let $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ be an SDBA, $Q_1, \delta_1, Q_2, \delta_2, \delta_t$ be the notation introduced in Figure 1, and $w = w_0 w_1 \ldots \in \Sigma^\omega$ be an infinite word. Each run ρ of $\mathcal{A}$ over w has one of the following properties:

1. ρ blocks,
2. ρ stays forever in Q_1,
3. ρ enters Q_2 and stops visiting F at some point, or
4. ρ is an accepting run.

Clearly, $w \notin L(\mathcal{A})$ if and only if every run of $\mathcal{A}$ over w has one of the first three properties. In the third case, we say that ρ is *safe* after visiting F for the last time (or since the moment it enters Q_2 if it does not visit any accepting state at all).

In order to check whether $w \in L(\mathcal{A})$ or not, one has to track all possible runs of $\mathcal{A}$. After reading a finite prefix of w, the states reached by the corresponding prefixes of runs can be divided into three sets.

1. The set $N \subseteq Q_1$ represents the runs that kept out of the deterministic part (N stands for *nondeterministic*) so far.
2. The set $C \subseteq Q_2$ represents the runs that have entered the deterministic part and that are not safe. One has to *check* (hence the name C) if some of them will be prolonged into accepting runs in the future, or if all of the runs eventually block or become safe.
3. The set $S \subseteq Q_2 \setminus F$ represents the *safe* runs.

Clearly, every accepting run of $\mathcal{A}$ stays in C after leaving N. On the other hand, if $w \notin L(\mathcal{A})$, every infinite run either stays in N or eventually leaves C to S and thus does not stay in C forever.

3.2 NCSB Complementation Construction

In this section, we describe an efficient construction that produces, for a given SDBA $\mathcal{A}$, a complement automaton $\mathcal{C}$. The automaton $\mathcal{C}$ has typically a low degree of non-determinism when compared to results of other complementation algorithms, and is always unambiguous. The complementation construction proposed here tracks the runs of $\mathcal{A}$ using the well known powerset construction and guesses the right classification of runs into sets N, C, and S. Moreover, in order to check that no run stays forever in C, it uses one more set $B \subseteq C$. The set B mimics the behaviour of C with one exception: it does not adopt the runs freshly coming to C via δ_t. The size of B never increases until it becomes empty; then we say that a *breakpoint* is reached. After each breakpoint, B is set to track exactly the runs currently in C. To sum up, states of $\mathcal{C}$ are quadruples (N, C, S, B)—hence the name NCSB complementation construction.

After reading only a finite prefix of the input word w, the automaton cannot know whether or not some run is already safe, as this depends on the suffix of w. The automaton $\mathcal{C}$ uses the guess-and-check strategy. Whenever a run ρ in C may freshly become safe (it is leaving an accepting state or it is entering Q_2 via a transit edge), then the automaton $\mathcal{C}$ makes a nondeterministic decision to move ρ to S or to leave it in C. The construction punishes every wrong decision:

- in order to preserve correctness, a run of $\mathcal{C}$ is blocked if ρ is moved to S too early (runs in S are not allowed to visit accepting states any more), and
- in order to maintain unambiguity, ρ is allowed to move from C to S only when leaving an accepting state. Hence, if ρ misses the moment when it leaves an accepting state for the last time, it will stay in C forever and this particular run of $\mathcal{C}$ cannot be accepting.

Before we formally describe the NCSB construction, we first naturally extend δ_1, δ_2, and δ_t to sets. For any $\bar{\delta} \in \{\delta_1, \delta_2, \delta_t\}$, any $a \in \Sigma$, and any set $X \subseteq Q_1$ or $X \subseteq Q_2$, we set $\bar{\delta}(X, a) = \bigcup_{q \in X} \bar{\delta}(q, a)$.

With the provided intuition in mind, we define the complement automaton NBA $\mathcal{C} = (P, \Sigma, \delta', I_{\mathcal{C}}, F_{\mathcal{C}})$ as follows.

- $P \subseteq 2^{Q_1} \times 2^{Q_2} \times 2^{Q_2 \setminus F} \times 2^{Q_2}$.
- $I_{\mathcal{C}} = \{(Q_1 \cap I, C, S, C) \mid S \cup C = I \cap Q_2, S \cap C = \emptyset\}$.
- $F_{\mathcal{C}} = \{(N, C, S, B) \in P \mid B = \emptyset\}$.
- δ' is the transition function $\delta' : P \times \Sigma \to 2^P$, such that $(N', C', S', B') \in \delta'\big((N, C, S, B), a\big)$ iff
  - $N' = \delta_1(N, a)$, $C' \cup S' = \delta_t(N, a) \cup \delta_2(C \cup S, a)$ (intuition: tracing the reachable states correctly),
  - $C' \cap S' = \emptyset$ (intuition: a run in Q_2 is either safe, or not),
  - $S' \supseteq \delta_2(S, a)$ (intuition: safe runs must stay safe),

- $C' \supseteq \delta_2(C \setminus F, a)$ (intuition: only runs leaving an accepting state can become safe),
- for all $q \in C \setminus F$, $\delta_2(q, a) \neq \emptyset$ (intuition: otherwise the corresponding run was safe already and should have been moved to S earlier), and
- if $B = \emptyset$ then $B' = C'$, and else $B' = \delta_2(B, a) \cap C'$ (intuition: breakpoint construction to check that no run stays in C forever).

Note that the only source of nondeterminism of δ' is when $\mathcal{C}$ has to guess correctly whether or not a run ρ of $\mathcal{A}$ is safe. Such situations arise in two cases, namely when the current state q of the run ρ satisfies

- $q \in \delta_t(N, a) \setminus (\delta_2(S, a) \cup F)$—$\rho$ is freshly entering Q_2, and when
- $q \in \delta_2(C \cap F, a) \setminus (\delta_2(S, a) \cup F)$—$\rho$ is leaving an accepting state.

All other situations are determined, including runs that are currently in $\delta_2(S, a)$ (which belong to S) and runs that are currently in F (which belong to C).

3.3 Complexity

Let $p = (N, C, S, B) \in P$ of $\mathcal{C}$. Then

- for a state $q_1 \in Q_1$ of $\mathcal{A}$, q_1 is either present or absent in N;
- for $q_2 \in F$, one of the following three options holds: q_2 is only in C, q_2 is both in C and B, or q_2 is not present in p at all; and
- for $q_3 \in Q_2 \setminus F$, one of the following four options holds: q_3 is only in S, q_3 is only in C, q_3 is both in C and B, or q_3 is not present in p at all.

The size of P is thus bounded by $|P| \leq 2^{|Q_1|} \cdot 3^{|F|} \cdot 4^{|Q_2 \setminus F|}$.

Let us note that, for deterministic automata (here we assume $\mathcal{A}$ is complete and Q_1 is empty), the NCSB construction leads to an automaton similar to an automaton with $2|Q| - |F|$ states produced by Kurshan's construction [18]. To see the size of the automaton produced by our construction for a DBA, recall that a state (N, C, S, B) of the complement automaton encodes that exactly the states in $N \cup C \cup S$ are reachable. For a DBA, $N \cup C \cup S$ thus contains exactly one state q of Q. Moreover, N is empty and thus B coincides with C since B becomes empty together with C. If $q \in F$, then it is in both B and C. If $q \in Q_2 \setminus F$, then it is either only in S, or in both B and C, leading to a size $\leq 2|Q_2| - |F|$.

3.4 Modification Suitable for On-the-fly Implementation

Some algorithms do not need to construct the whole complement automaton. For example, in order to verify that $w \notin L(\mathcal{A})$ one only needs to built the accepting lasso in $\mathcal{C}$ for w. Or when building a product with some other automaton (or Markov chain), it is unnecessary to build the part of $\mathcal{C}$ which is not used in the product. Further, some tools work with implicitly encoded automata and/or query an SMT solver to check the presence of a transition in the automaton, which is expensive. Ultimate Büchi Automizer has both properties: it stores automata in an implicit form and builds a product of the complement with a program flowgraph. Such tools can greatly benefit from an *on-the-fly* complementation that does not rely on the knowledge of the whole input automaton.

Our complementation can be easily adapted for an on-the-fly implementation. Because we have no knowledge about Q_1, Q_2, and δ_t in this variation, the runs are held in N until they reach an accepting state, only then they are moved to C.

Technically, the "$N' = \delta_1(N, a)$" from the definition of δ' would be replaced by "$N' = \delta(N, a) \smallsetminus F$" and for C' now holds $C' \subseteq \delta(C, a) \cup (\delta(N, a) \cap F)$. The on-the-fly construction can therefore have up to $2^{|Q_1|} \cdot 3^{|F|} \cdot 5^{|Q_2 \smallsetminus F|}$ states.

Note that the on-the-fly construction does not add any further nondeterminism to the construction. To the contrary, there is an injection of runs from the construction discussed in Section 3.2 to this on-the-fly construction. The correctness argument and the uniqueness argument for the accepting run which are given in Section 4 therefore need only very minor adjustments.

4 Level Rankings in Complementation and Correctness

We open this section by introduction of *run graphs* and *level rankings*. We then look at our construction through the level ranking lense and use the insights this provides for proving its correctness and unambiguity.

4.1 Complementation and Level Rankings

In [17], Kupferman and Vardi introduce *level rankings* as a witness for the absence of accepting runs of Büchi automata. They form the foundation of several complementation algorithms [17,13,12,27,29].

The set of all runs of a nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ over a word w can be represented by a directed acyclic graph $\mathcal{G}_w = (V, E)$, called the *run graph* of $\mathcal{A}$ on w, with

- vertices $V \subseteq Q \times \omega$ such that $(q, i) \in V$ iff there is a run $\rho = q_0 q_1 q_2 \dots$ over $\mathcal{A}$ on w with $q_i = q$, and
- edges $E \subseteq (Q \times \omega) \times (Q \times \omega)$ such that $\big((q, i), (q', i')\big) \in E$ iff $i' = i + 1$ and there is a run $\rho = q_0 q_1 q_2 \dots$ of $\mathcal{A}$ over w with $q_i = q$ and $q_{i+1} = q'$.

The run graph $\mathcal{G}_w$ is called *rejecting* if no path in $\mathcal{G}_w$ satisfies the Büchi condition. That is, $\mathcal{G}_w$ is rejecting iff w does not have any accepting run, and thus iff w is not in the language of $\mathcal{A}$. $\mathcal{A}$ can be complemented to a nondeterministic Büchi automaton $\mathcal{C}$ that checks if $\mathcal{G}_w$ is rejecting.

The property that $\mathcal{G}_w$ is rejecting can be expressed in terms of *ranks* [17]. We call a vertex $(q, i) \in V$ of a graph $\mathcal{G} = (V, E)$ *safe*, if no vertex reachable from (q, i) is accepting (that is, in $F \times \omega$), and *finite*, if the set of vertices reachable from (q, i) in $\mathcal{G}$ is finite.

Based on these definitions, *ranks* can be assigned to the vertices of a rejecting run graph. We set ${\mathcal{G}_w}^0 = \mathcal{G}_w$, and repeat the following procedure until a fixed point is reached, starting with $i = 1$:

- Assign all safe vertices of ${\mathcal{G}_w}^{i-1}$ the rank i, and set ${\mathcal{G}_w}^{i}$ to ${\mathcal{G}_w}^{i-1}$ minus the vertices with rank i (that is, minus the safe vertices in ${\mathcal{G}_w}^{i-1}$).
- Assign all finite vertices of ${\mathcal{G}_w}^{i}$ the rank $i+1$, and set ${\mathcal{G}_w}^{i+1}$ to ${\mathcal{G}_w}^{i}$ minus the vertices with rank $i+1$ (that is, minus the finite vertices in ${\mathcal{G}_w}^{i}$).
- Increase i by 2.

A fixed point is reached in $n+2$ steps[2], and the ranks can be used to characterise the complement language of a nondeterministic Büchi automaton:

Proposition 1. *[17] A nondeterministic Büchi automaton $\mathcal{A}$ with n states rejects a word w iff ${\mathcal{G}_w}^{2n+2}$ is empty.* □

4.2 Ranks and Complementation of SDBAs

When considering the run graph for SBDAs, we only need to consider three ranks: 1, 2, and 3. What is more, the vertices $Q_2 \times \omega$ reachable from accepting vertices can only have rank 1 or rank 2 in a rejecting run graph.

Proposition 2. *A semi-deterministic Büchi automaton $\mathcal{A}$ rejects a word w iff ${\mathcal{G}_w}^{3}$ is empty. This is the case iff ${\mathcal{G}_w}^{2}$ contains no vertex in $Q_2 \times \omega$.*

Proof. Let w be a word rejected by $\mathcal{S}$. By construction, ${\mathcal{G}_w}^{1}$ contains no safe vertices. (Note that removing safe vertices does not introduce new safe vertices.)

Let us assume for contradiction that ${\mathcal{G}_w}^{1}$ contains a vertex $(q_i, i) \in Q_2 \times \omega$, which is not finite. As (q_i, i) is not finite, there is an infinite run $\rho = q_0 q_1 q_2 \ldots q_{i-1} q_i q_{i+1} \ldots$ of $\mathcal{A}$ over w such that, for all $j \geq i$, (q_j, j) is a vertex in ${\mathcal{G}_w}^{1}$. This is because $q_i \in Q_2$, the deterministic part of the SBDA, and $\{(q_j, j) \mid j \geq i\}$ is therefore (1) determined by w and (q_i, i), and (2) fully in ${\mathcal{G}_w}^{1}$, because otherwise (q_i, i) would be finite.

But if all vertices in $\{(q_j, j) \mid j \geq i\}$ are in ${\mathcal{G}_w}^{1}$, then none of them is safe in $\mathcal{G}_w$. Using again that the tail $q_i q_{i+1} q_{i+2} \ldots$ is unique and well defined (as $q_i \in Q_2$, the deterministic part of the SDBA), it follows that, for all $j \geq i$, there is a $k \geq j$ such that q_k is accepting. Consequently, ρ is accepting (contradiction).

We have thus shown that, if $\mathcal{S}$ rejects a word w, then ${\mathcal{G}_w}^{2}$ contains no state in $Q_2 \times \omega$. This also implies that ${\mathcal{G}_w}^{2}$ contains no accepting vertices. Consequently, all vertices in ${\mathcal{G}_w}^{2}$ are safe. Consequently, ${\mathcal{G}_w}^{3}$ is empty. □

We now consider the NCSB construction from a level ranking perspective. We start with an intuition for the *rational* run $\rho = (N_0, C_0, S_0, B_0)(N_1, C_1, S_1, B_1)(N_2, C_2, S_2, B_2) \ldots$ of $\mathcal{C}$ over a word w rejected by $\mathcal{A}$, where $(V, E) = \mathcal{G}_w$. A rational run is the unique accepting run of $\mathcal{C}$ over w and it guesses the ranks precisely, that is:

[2] It is common to use 0 as the minimal rank (i.e. to start with the finite vertices), but the correctness of the complementation does not rely on this. The proof in [17] refers to this case, and requires $n+1$ steps. For our purpose, the minimal rank needs to be odd, i.e. we need to start with safe vertices.

- $N_i = \{q \mid (q,i) \in V, q \in Q_1\}$,
- $C_i = \{q \mid (q,i) \in V, q \in Q_2$ and the rank of (q,i) is $2\}$ (we need to check that these states are finite in ${\mathcal{G}_w}^2$),
- $S_i = \{q \mid (q,i) \in V, q \in Q_2$ and the rank of (q,i) is $1\}$,
- $B_i \subseteq C_i$.

All runs of $\mathcal{C}$ that differ on some i from the rational run will either block or will keep the wrongly guessed vertices with rank 1 in C and thus will be not accepting.

Note that the $\mathcal{C}$ does not need to guess much. The development of the N_i is deterministic. The development of $C_i \cup S_i$ is deterministic, S_i and C_i are disjoint, and states in F cannot be in S_i. The B_i serve as a breakpoint construction, and the development of B_i is determined by the development of the C_i. All that needs to be guessed is the point when a vertex becomes safe, and there is only a single correct guess.

4.3 Correctness

After reading only a finite prefix of an input word w, the automaton has to use its nondeterministic power to guess which reached state in Q_2 should be added to S. We now establish that the automaton $\mathcal{C}$ is an unambiguous automaton that recognises the complement language of $\mathcal{A}$ by showing

1. $\mathcal{C}$ does not accept a word that is accepted by $\mathcal{A}$,
2. for words that are not accepted by $\mathcal{A}$, the run inferred from the level ranking discussed in Section 4.2 defines an accepting run, and
3. for words w that are not accepted by $\mathcal{A}$, this is the only accepting run of $\mathcal{C}$ over w.

Lemma 1. *Let $\mathcal{A}$ be an SDBA, $\mathcal{C}$ be constructed by the NCSB complementation of $\mathcal{A}$, and $w \in L(\mathcal{A})$ be a word in the language of $\mathcal{A}$. Then $\mathcal{C}$ does not accept w.*

Proof. Let $\rho = q_0 q_1 \ldots$ be an accepting run of $\mathcal{A}$ over w, and let $i \in \omega$ be an index such that $q_i \in F$. Let us assume for contradiction that $\rho' = (N_0, C_0, S_0, B_0)(N_1, C_1, S_1, B_1) \ldots (N_n, C_n, S_n, B_n) \ldots$ is an accepting run of $\mathcal{C}$ over w. Clearly, $q_i \in C_i$. It therefore holds, for all $j \geq i$, that $q_j \in C_j \cup S_j$.

We look at the following case distinction.

1. For all $j \geq i$, $q_j \in C_j$. As ρ' is accepting, there is a breakpoint ($B_j = \emptyset$) for some $j \geq i$. For such a j we have that $q_{j+1} \in B_{j+1}$ and, moreover, that $q_k \in B_k$ for all $k \geq j+1$. Thus, $B_k \neq \emptyset$ for all $k \geq j+1$ and ρ' visits only finitely many accepting states (contradiction).
2. There is a $j \geq i$ such that $q_j \in S_j$. But then $q_k \in S_k$ holds for all $k \geq j$ by construction. However, as ρ is accepting, there is an $l \geq j$ such that $q_l \in F$, which contradicts $q_l \in S_l$ (contradiction).

□

Lemma 2. *Let $\mathcal{A}$ be an SDBA, $\mathcal{C}$ be the automaton constructed by the NCSB complementation of $\mathcal{A}$, $w \notin L(\mathcal{A})$, and $(V, E) = \mathcal{G}_w$ be the run graph of $\mathcal{A}$ on w. Then there is exactly one* rational *run of the form $\rho = (N_0, C_0, S_0, B_0)(N_1, C_1, S_1, B_1)(N_2, C_2, S_2, B_2)\ldots$. This run is accepting.*

Proof. It is easy to check that this defines exactly one infinite run: the updates of the N, C, and S components follow the rules for transitions from the definition of $\mathcal{C}$, and the update of the B component is fully determined by the update of C and the previous value of B.

What remains is to show that the run is accepting. Let us assume for contradiction that there are only finitely many breakpoints reached, i.e. there is an index $i \in \omega$, for which there is no $j \geq i$, such that $B_j = \emptyset$.

Now we have $\emptyset \neq B_i \subseteq C_i = \{q \mid (q, i) \in V \text{ s.t. } q \in Q_2$ and the rank of (q, i) is $2\}$. The construction provides that, if there is no breakpoint on or after position i, then B_j is the set of states that correspond to vertices from $Q \times \{j\}$ reachable in ${\mathcal{G}_w}^1$ from the vertices $B_i \times \{i\}$. As there is no future breakpoint, there are infinitely many such vertices, and Königs lemma implies that there is an infinite path in ${\mathcal{G}_w}^1$ from at least one of the vertices in $B_i \times \{i\}$. This provides a contradiction to the assumption that the rank of these vertices is 2, i.e. that they are finite in ${\mathcal{G}_w}^1$. □

Lemma 3. *Let $\mathcal{A}$ be an SDBA, $\mathcal{C}$ be the automaton constructed by the NCSB complementation of $\mathcal{A}$, $w \notin L(\mathcal{A})$, and $(V, E) = \mathcal{G}_w$ be the run graph of $\mathcal{A}$ on w. Let $\rho = (N_0, C_0, S_0, B_0)(N_1, C_1, S_1, B_1)(N_2, C_2, S_2, B_2)\ldots$ be an infinite, non-rational run of $\mathcal{C}$ over w that is, it does* not *satisfy*

- $N_i = \{q \mid (q, i) \in V \text{ s.t. } q \in Q_1\}$,
- $C_i = \{q \mid (q, i) \in V \text{ s.t. } q \in Q_2 \text{ and the rank of } (q, i) \text{ is } 2\}$,
- $S_i = \{q \mid (q, i) \in V \text{ s.t. } q \in Q_2 \text{ and the rank of } (q, i) \text{ is } 1\}$,

for some i. Then ρ is rejecting.

Proof. As the N part always tracks the reachable states in Q_1 correctly by construction, and the $C \cup S$ part always tracks the reachable states in Q_2 correctly by construction, we have one of the following two cases according to Proposition 2.

The first case is that there is a safe vertex $(q, i) \in V$ such that $q \in C_i$. By construction, a unique maximal path $(q_i, i)(q_{i+1}, i+1)(q_{i+2}, i+2)(q_{i+3}, i+3)\ldots$ for $q_i = q$ exists in $\mathcal{G}_w$, and this path does not contain any accepting state. By an inductive argument, for all vertices (q_j, j) on this path, $q_j \in C_j$. If the path is finite, ρ blocks at the end (due to the definition of the transition function of $\mathcal{C}$), which contradicts the assumption that the run ρ is infinite. Similarly, if the path is infinite, $q_k \in B_k$ for some $k \geq i$. Then $q_j \in B_j$ for all $j > k$ with (q_j, j) on this path. Therefore, ρ cannot be accepting.

The second case is that there is a non-safe vertex in $(q, i) \in V$ such that $q \in S_i$. (Note that this implies $q \notin F$.) By construction, we get, for $q_i = q$, a unique maximal path $(q_i, i)(q_{i+1}, i+1)(q_{i+2}, i+2)(q_{i+3}, i+3)\ldots$ in $\mathcal{G}_w$, and this path contains an accepting state q_k. By an inductive argument, for all vertices (q_j, j) on this path, $q_j \in S_j$. But this implies $q_k \in S_k$ (contradiction). □

The first two lemmas provide the correctness of our complementation algorithm. Considering that no finite run is accepting, the third lemma establishes that $\mathcal{C}$ is unambiguous.

Theorem 1. *Let $\mathcal{A}$ be an SDBA and $\mathcal{C}$ be the automaton constructed by the NCSB complementation of $\mathcal{A}$. Then $\mathcal{C}$ is an unambiguous Büchi automaton that recognises the complement of the language of $\mathcal{A}$.*

5 Experimental Evaulation

This section compares the results of the NCSB complementation with these produced by well-known complementations for nondeterministic Büchi automata. All the automata, tools, scripts and commands used in the evaluation, and some further comparisons can be found at `https://github.com/xblahoud/NCSB-Complementation`.

5.1 Implementations of the NCSB Complementation

We implemented the NCSB complementation in two tools. One implementation is available in the Goal tool[3] [33]. Goal is a graphical interactive tool for omega automata, temporal logics, and games. It provides several Büchi complementation algorithms and was used in an extensive evaluation of these algorithms [32]. In the commandline version, the parameter for our construction is `complement -m sdbw -a`. The partition of the set Q into Q_1 and Q_2 is not a parameter, instead the implementation uses the set of all states that are reachable from some accepting state as Q_2.

Our second implementation is available in the Ultimate Automata Library. This library is used by the termination analyser Ultimate Büchi Automizer and other tools of the Ultimate program analysis framework[4]. The implementation uses the on-the-fly construction discussed in Section 3.4. The library provides a language that allows users to define automata and a sequence of commands that should be executed by the library. This language is called *automata script* and an interpreter for this language is available via a web interface[5]. The operation that implements the NCSB construction has the name `buchiComplementNCSB`.

5.2 Example Automata

For our evaluation, we took automata whose complementation was a subtask while the tool Ultimate Büchi Automizer was analysing the programs from the Termination category of the software verification competition SV-COMP 2015 [4]. We wrote each Büchi automaton that was semi-deterministic but not

[3] `http://goal.im.ntu.edu.tw/`

[4] `http://ultimate.informatik.uni-freiburg.de/`

[5] `http://ultimate.informatik.uni-freiburg.de/automata_script_interpreter`

deterministic to a file in the Hanoi omega-automata format [2]. We obtained 106 semi-deterministic Büchi automata. Using the command `autfilt --unique -H` from the Spot library [10], we identified isomorphic automata and kept only the remaining 97 pairwise non-isomorphic ones.

By construction, all these automata behave deterministically only after the first visit of an accepting state. Hence the partition of the states Q into Q_1 and Q_2 is unique and the results of the construction presented in Section 3.2 and the results of the on-the-fly modification presented in Section 3.4 coincide.

5.3 Other Complementation Constructions

The known constructions for the complementation of nondeterministic Büchi automata can be classified into the following four categories.

Ramsey-based. Historically the first complementation construction introduced by Büchi [6] and later improved by Sistla, Vardi, and Wolper [31] in which a Ramsey-based combinatorial argument is involved.

Determinisation-based. A construction proposed by Safra [25] and later enhanced by Piterman [24] in which a state of a complement is represented by a Safra tree.

Rank-based. A construction introduced by Kupferman and Vardi [17] for which several optimisations [17,13,12,27] have been proposed.

Slice-based. A construction [16] proposed by Kähler and Wilke that constructs complements accepting reduced split trees rather than run graphs.

For each of these categories, GOAL provides implementations that can be adjusted by various parameters. In our evaluation, we included one construction from each category. For the latter three categories, we took the arguments that were most successful in an extensive evaluation [32]. For the first category, we used additionally an optimisation that minimises the finite automata that are constructed during the complementation [5]. The commands that we used are listed in Table 1.

Table 1. Complementation constructions of NBAs used in our evaluation

construction	GOAL command
Ramsey-based	`complement -m ramsey -macc -min`
Determinisation-based	`complement -m piterman -macc -sim -eq`
Rank-based	`complement -m rank -macc -tr -ro -cp`
Slice-based	`complement -m slice -macc -eg -madj -ro`

Table 2. Results of complementation constructions without posteriori simplifications

construction	91 easy SDBAs		6 difficult SDBAs					
	states	transitions	1	2	3	4	5	6
Ramsey-based	16909	848969	–	–	–	–	–	–
Rank-based	2703	21095	–	–	1022	7460	8245	–
Det.-based	1841	24964	–	–	172	346	385	3527
Slice-based	1392	14783	66368	–	184	421	475	9596
NCSB	950	8003	20711	84567	108	343	401	5449

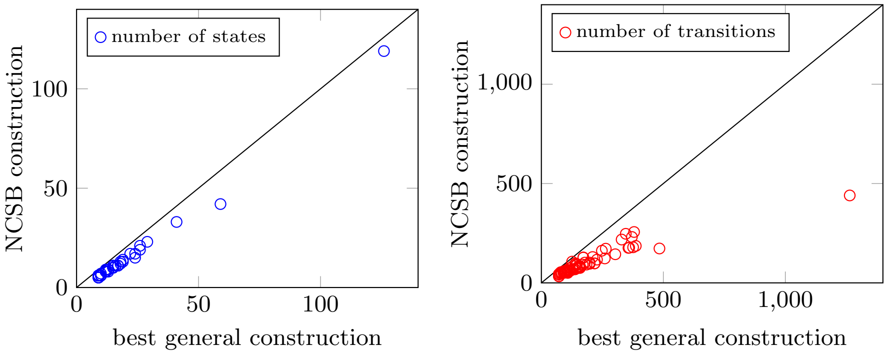

Fig. 2. Comparison of the NCSB construction and other complementations

5.4 Evaluation

We applied the NCSB complementation and the four complementations of Table 1 to the 97 pairwise non-isomorphic SDBAs. All complementations were run on a laptop with an Intel Core i5 2.70GHz CPU. We restricted the maximal heap space of the JVM to 8GB (all complementations are implemented in Java) and used a timeout of 300s. The results are depicted in Table 2 and Figure 2.

For 91 out of 97 SDBAs, all implementations were able to compute a result. We refer to these 91 SDBAs as easy SDBAs, while the remaining six are referenced as difficult in the Table 2. For each complementation, we provide the cumulative numbers of states and transitions of all 91 easy complements. For each of the easy SDBAs, NCSB construction produces the complement with the smallest number of states. In Figure 2, a size of the complement produced by the NCSB construction is compared to the size of the smallest complement produced by the constructions of Table 1 for each of the easy automata.

For the difficult SDBAs, at least one construction was not able to provide the result within the given time and memory limits. We provide the number of states of the computed complements for each of them. While there are two cases where the determinisation-based construction produced an automaton with less

Table 3. Complementations and simplifications

construction	no simplifications		with simplifications			failure	
	states	transitions	states	transitions	min	compl.	simp.
GoalRamsey	6386	172351	5223	90548	0	6	22
GoalRank	1437	11677	899	7657	4	3	14
GoalSafraPiterman	1300	15491	1083	9589	0	2	11
GoalSlice	892	8921	785	6789	4	1	13
UltimateBS	598	4922	514	4460	73	0	10

states than the NCSB construction, the number of transition was always smaller for the NCSB construction.

A common approach to mitigate the problem of large complementation results is to apply generic size reduction algorithm. Does our NCSB construction also outperform the other constructions if we apply size reduction techniques afterwards? In order to address this question, we applied the "simplification routines" of the SPOT library [1] (in version 1.99.4a) to the complements. We run the command `autfilt --small --high -B -H` with a timeout of 300s and obtained the results depicted in Table 3. For 75 SDBAs, all complements could be simplified within the timeout. For these we again provide the cumulative numbers of states and transitions before and after the simplifications. The column *min* shows how often each construction followed by simplification produced a complement with the minimal number of states. The column *failure* shows how often a timeout prevented a successful complementation or simplification. It is interesting to see that the simplifications were not able to reduce the number of transitions much for the NCSB construction, while they were able to reduce it by more than 20% in case of the other complementations.

6 Conclusion

We have introduced an efficient complementation construction for semi-deterministic Büchi automata (SDBA). The results of our construction have two appealing properties: they are unambiguous and have less than 4^n states. We have presented a modification of our construction suitable for implementation on-the-fly and showed that our construction can be seen as a specialised version of the rank-based construction for nondeterministic Büchi automata. We have implemented our construction in two tools and did an experimental evaluation on semi-deterministic Büchi automata produced by the termination analyser ULTIMATE BÜCHI AUTOMIZER. We have compared our construction to four known complementation constructions for (general) nondeterministic Büchi automata. The evaluation showed that our construction outperforms the existing constructions in the number of states and transitions.

References

1. T. Babiak, T. Badie, A. Duret-Lutz, M. Křetínský, and J. Strejček. Compositional approach to suspension and other improvements to LTL translation. In *SPIN 2013*, volume 7976 of *LNCS*, pages 81–98. Springer, 2013.
2. T. Babiak, F. Blahoudek, A. Duret-Lutz, J. Klein, J. Křetínský, D. Müller, D. Parker, and J. Strejček. The Hanoi omega-automata format. In *CAV 2015*, volume 9206 of *LNCS*, pages 479–486. Springer, 2015.
3. M. Benedikt, R. Lenhardt, and J. Worrell. LTL model checking of interval Markov chains. In *TACAS 2013*, volume 7795 of *LNCS*, pages 32–46. Springer, 2013.
4. D. Beyer. Software verification and verifiable witnesses - (report on SV-COMP 2015). In *TACAS 2015*, volume 9035 of *LNCS*, pages 401–416. Springer, 2015.
5. S. Breuers, C. Löding, and J. Olschewski. Improved Ramsey-based Büchi complementation. In *FoSSaCS 2012*, volume 7213 of *LNCS*, pages 150–164. Springer, 2012.
6. J. R. Büchi. On a decision method in restricted second order arithmetic. In *CLMPS 1960*, pages 1–11. Stanford University Press, 1962.
7. D. Bustan, S. Rubin, and M. Y. Vardi. Verifying ω-regular properties of Markov chains. In *CAV 2004*, volume 3114 of *LNCS*, pages 189–201. Springer, 2004.
8. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.
9. D. Dietsch, M. Heizmann, V. Langenfeld, and A. Podelski. Fairness modulo theory: A new approach to LTL software model checking. In *CAV 2015*, volume 9206 of *LNCS*, pages 49–66. Springer, 2015.
10. A. Duret-Lutz and D. Poitrenaud. SPOT: An extensible model checking library using transition-based generalized Büchi automata. In *MASCOTS 2004*, pages 76–83. IEEE Computer Society, 2004.
11. S. Fogarty, O. Kupferman, T. Wilke, and M. Y. Vardi. Unifying Büchi complementation constructions. *Logical Methods in Computer Science*, 9(1), 2013.
12. E. Friedgut, O. Kupferman, and M. Y. Vardi. Büchi complementation made tighter. *International Journal of Foundations of Computer Science*, 17(4):851–868, 2006.
13. S. Gurumurthy, O. Kupferman, F. Somenzi, and M. Y. Vardi. On complementing nondeterministic Büchi automata. In *CHARME 2003*, volume 2860 of *LNCS*, pages 96–110. Springer, 2003.
14. E. M. Hahn, G. Li, S. Schewe, A. Turrini, and L. Zhang. Lazy probabilistic model checking without determinisation. In *CONCUR 2015*, volume 42 of *LIPIcs*, pages 354–367. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.
15. M. Heizmann, J. Hoenicke, and A. Podelski. Termination analysis by learning terminating programs. In *CAV 2014*, volume 8559 of *LNCS*, pages 797–813. Springer, 2014.
16. D. Kähler and T. Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *ICALP 2008*, volume 5125 of *LNCS*, pages 724–735. Springer, 2008.
17. O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(2):408–429, July 2001.
18. R. P. Kurshan. *Computer-aided verification of coordinating processes: the automata-theoretic approach.* Princeton University Press, 1994.
19. R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, October 1966.

20. M. Michel. Complementation is more difficult with automata on infinite words. Technical report, CNET, Paris (Manuscript), 1988.
21. S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32(3):321–330, 1984.
22. D. E. Muller. Infinite sequences and finite machines. In *FOCS 1963*, pages 3–16. IEEE Computer Society Press, 1963.
23. J.-P. Pécuchet. On the complementation of Büchi automata. *Theoretical Computer Science*, 47(3):95–98, 1986.
24. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007.
25. S. Safra. On the complexity of omega-automata. In *FOCS 1988*, pages 319–327. IEEE Computer Society, 1988.
26. W. J. Sakoda and M. Sipser. Non-determinism and the size of two-way automata. In *STOC 1978*, pages 274–286. ACM Press, 1978.
27. S. Schewe. Büchi complementation made tight. In *STACS 2009*, volume 3 of *LIPIcs*, pages 661–672. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.
28. S. Schewe. Tighter bounds for the determinisation of Büchi automata. In *FOSSACS 2009*, volume 5504 of *LNCS*, pages 167–181. Springer, 2009.
29. S. Schewe and T. Varghese. Tight bounds for the determinisation and complementation of generalised Büchi automata. In *ATVA 2012*, volume 7561 of *LNCS*, pages 42–56. Springer, 2012.
30. S. Schewe and T. Varghese. Tight bounds for complementing parity automata. In *MFCS 2014*, volume 8634 of *LNCS*, pages 499–510. Springer, 2014.
31. A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(3):217–239, 1987.
32. M. Tsai, S. Fogarty, M. Y. Vardi, and Y. Tsay. State of Büchi complementation. *Logical Methods in Computer Science*, 10(4), 2014.
33. M. Tsai, Y. Tsay, and Y. Hwang. GOAL for games, omega-automata, and logics. In *CAV 2013*, volume 8044 of *LNCS*, pages 883–889. Springer, 2013.
34. M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS 1985*, pages 327–338. IEEE Computer Society, 1985.
35. M. Y. Vardi. The Büchi complementation saga. In *STACS 2007*, volume 4393 of *LNCS*, pages 12–22. Springer, 2007.
36. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS 1986*, pages 332–344. IEEE Computer Society, 1986.
37. Q. Yan. Lower bounds for complementation of *omega*-automata via the full automata technique. *Logical Methods in Computer Science*, 4(1:5), 2008.