State of Büchi Complementation*

Ming-Hsien Tsai[1], Seth Fogarty[2], Moshe Y. Vardi[2], and Yih-Kuen Tsay[1]

[1] National Taiwan University
[2] Rice University

Abstract. Büchi complementation has been studied for five decades since the formalism was introduced in 1960. Known complementation constructions can be classified into Ramsey-based, determinization-based, rank-based, and slice-based approaches. For the performance of these approaches, there have been several complexity analyses but very few experimental results. What especially lacks is a comparative experiment on all the four approaches to see how they perform in practice. In this paper, we review the state of Büchi complementation, propose several optimization heuristics, and perform comparative experimentation on the four approaches. The experimental results show that the determinization-based Safra-Piterman construction outperforms the other three and our heuristics substantially improve the Safra-Piterman construction and the slice-based construction.

1 Introduction

Büchi automata are nondeterministic finite automata on infinite words that recognize ω-regular languages. It is known that Büchi automata are closed under Boolean operations, namely union, intersection, and complementation. Complementation was first studied by Büchi in 1960 for a decision procedure for second-order logic [3]. Complementation of Büchi automata is significantly more complicated than that of nondeterministic finite automata on finite words. Given a nondeterministic finite automaton on finite words with n states, complementation yields an automaton with 2^n states through the subset construction. Indeed, for nondeterministic Büchi automata, the subset construction is insufficient for complementation. In fact, Michel showed in 1988 that blow-up of Büchi complementation is at least $n!$ (approximately $(n/e)^n$ or $(0.36n)^n$), which is much higher than 2^n [17]. This lower bound was later sharpened by Yan to $(0.76n)^n$ [31], which was matched by an upper bound by Schewe [21].

There are several applications of Büchi complementation in formal verification, for example, verifying whether a system satisfies a property by checking if the intersection of the system automaton and the complement of the property automaton is empty [27], testing the correctness of an LTL translation algorithm without a reference algorithm, etc. [9]. Although recently many works focus on

* Work supported in part by the National Science Council, Taiwan (R.O.C.) under grant NSC97-2221-E-002-074-MY3, by NSF grants CCF-0613889, ANI-0216467, CCF-0728882, and OISE-0913807, by BSF grant 9800096, and by gift from Intel.

M. Domaratzki and K. Salomaa (Eds.): CIAA 2010, LNCS 6482, pp. 261–271, 2011.

universality and containment testing without going explicitly through complementation [5,6,4], it is still unavoidable in some cases [16].

Known complementation constructions can be classified into four approaches: Ramsey-based approach [3,22], determinization-based approach [20,18,2,19], rank-based approach [24,15,13], and slice-based approach [10,30]. The first three approaches were reviewed in [29]. Due to the high complexity of Büchi complementaton, optimization heuristics are critical to good performance [9,7,21,11,14]. Unlike the rich theoretical development, empirical studies of Büchi complementation have been rather few [14,9,11,26], as much recent emphasis has shifted to universality and containment. A comprehensive empirical study would allow us to evaluate the performance of these complementation approaches.

Vardi, The Büchi Complementation Saga

In this paper, we review the four complementation approaches and perform comparative experimentation on the best construction in each approach. Although the conventional wisdom is that the nondeterministic constructions are better than the deterministic construction, due to better worse-case bounds, the experimental results show that the deterministic construction is the best for complementation in general. At the same time, the Ramsey-based approach, which is competitive in universality and containment testing [1,5,6], performs rather poorly in our complementation experiments. We also propose optimization heuristics for the determinization-based construction, the rank-based construction, and the slice-based construction. The experiment shows that the optimization heuristics substantially improve the three constructions. Overall, our work confirms the importance of experimentation and heuristics in studying Büchi complementation, as worst-case bounds are poor guides to actual performance.

This paper is organized as follows. Some preliminaries are given in Section 2. In Section 3, we review the four complementation approaches. We discuss the results of our comparative experimentation on the four approaches in Section 4. Section 5 describes our optimization heuristics and Section 6 shows the improvement made by our heuristics. We conclude in Section 7. More results of the experiments in Section 4 and Section 6 and further technical details regarding some of the heuristics can be found in [25].

2 Preliminaries

A *nondeterministic* ω-automaton A is a tuple $(\Sigma, Q, q_0, \delta, \mathcal{F})$, where Σ is the finite alphabet, Q is the finite state set, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $\mathcal{F}$ is the acceptance condition, to be described subsequently. A is *deterministic* if $|\delta(q, a)| = 1$ for all $q \in Q$ and $a \in \Sigma$.

all infinite words over alphabet sigma

Given an ω-automaton $A = (\Sigma, Q, q_0, \delta, \mathcal{F})$ and an infinite word $w = a_0 a_1 \cdots \in \Sigma^\omega$, a *run* ρ of A on w is a sequence $q_0 q_1 \cdots \in Q^\omega$ satisfying $\forall i : q_{i+1} \in \delta(q_i, a_i)$. A run is *accepting* if it satisfies the acceptance condition. A word is *accepted* if there is an accepting run on it. The *language* of an ω-automaton A, denoted by $L(A)$, is the set of words accepted by A. An ω-automaton A is *universal* if $L(A) = \Sigma^\omega$. A state is *live* if it occurs in an accepting run on some word, and is *dead* otherwise. Dead states can be discovered using a nonemptiness algorithm, cf. [28], and can be pruned off without affecting the language of the automaton.

automaton accepts all possible infinite words

Difference between Büchi automata, Muller automata, Rabin automata, etc: acceptance condition.

Let ρ be a run and $inf(\rho)$ be the set of states that occur infinitely often in ρ. Various ω-automata can be defined by assigning different acceptance conditions: *Büchi condition* where $\mathcal{F} \subseteq Q$ and ρ satisfies the condition iff $inf(\rho) \cap \mathcal{F} \neq \emptyset$; *Rabin condition* where $\mathcal{F} \subseteq 2^Q \times 2^Q$ and ρ satisfies the condition iff there exists $(E, F) \in \mathcal{F}$ such that $inf(\rho) \cap E = \emptyset$ and $inf(\rho) \cap F \neq \emptyset$; *parity condition* where $\mathcal{F} : Q \to \{0, 1, \ldots, 2r\}$ and ρ satisfies the condition iff $min\{\mathcal{F}(q) \mid q \in inf(\rho)\}$ is even. $\mathcal{F}(q)$ is called the parity of a state q.

We use a system of three-letter acronyms to denote these ω-automata. The first letter indicates whether the automaton is **n**ondeterministic or **d**eterministic. The second letter indicates whether the acceptance condition is **B**üchi, **R**abin, or **p**arity. The third letter is always a "**W**" indicating the automaton accepts words. For example, NBW stands for a nondeterministic Büchi automaton and DPW stands for a deterministic parity automaton.

Given an ω-automaton A and an infinite word w, the *run tree* of A on w is a tree where the vertices of a (full) branch form a run of A on w and there is a corresponding branch for every run of A on w. The *split tree* of A on w is a binary tree that abstracts the run tree by grouping accepting successors and nonaccepting successors of states in a vertex respectively into the left child and the right child. The *reduced split tree* of A on w is a binary tree obtained from the split tree of A on w by removing a state from a vertex if it also occurs in a vertex to the left on the same level and removing a vertex if it contains no state. An NBW accepts a word if there is a left-recurring branch in the reduced split tree. A *slice* is a sequence of state sets representing all vertices on a same level of a reduced split tree in an order from left to right.

deletion from left to right instead of right to left

3 Historical Review

Ramsey-based approach. The very first complementation construction introduced by Büchi in 1960 involves a Ramsey-based combinatorial argument and results in a $2^{2^{O(n)}}$ blow-up in the state size [3]. This construction was later improved by Sistla, Vardi, and Wolper to reach a single-exponential complexity $2^{O(n^2)}$ [22]. In the improved construction, referred to as `Ramsey` in this paper, the complement is obtained by composing certain automata among a set of Büchi automata which form a partition of Σ^ω, based on Ramsey's Theorem. Various optimization heuristics for the Ramsey-based approach are described in [1,6], but the focus in these works is on universality and containment. In spite of the quadratic exponent of the Ramsey-based approach, it is shown in [1,5,6] to be quite competitive for universality and containment.

Determinization-based approach. Safra's $2^{O(n \log n)}$ construction is the first complementation construction that matches the $\Omega(n!)$ lower bound [20]. Later on, Muller and Schupp introduced a similar determinization construction which records more information and yields larger complements in most cases, but can be understood more easily [18,2]. In [19], Piterman improved Safra's construction by using a more compact structure and using parity automata as the intermediate deterministic automata, which yields an upper bound of n^{2n}. Piterman's

construction, referred to as `Safra-Piterman` in this paper, performs complementation in stages: from NBW to DPW, from DPW to complement DPW, and finally from complement DPW to complement NBW. The idea is the use of (1) a compact Safra tree to capture the history of all runs on a word and (2) marks to indicate whether a run passes an accepting state again or dies.

Since the determinization-based approach performs complementation in stages, different optimization techniques can be applied separately to the different stages. For instance, several optimization heuristics on Safra's determinization and on simplifying the intermediate DRW were proposed by Klein and Baier [14].

Rank-based approach. The rank-based approach, proposed by Kupferman and Vardi, uses rank functions to measure the progress made by a node of a run tree towards fair termination [15]. The basic idea of this approach may be traced back to Klarlund's construction with a more complex measure [13]. Both constructions have complexity $2^{O(n \log n)}$. There were also several optimization techniques proposed in [9,7,11]. A final improvement was proposed recently by Schewe [21] to the construction in [7]. The later construction performs a subset construction in the first phase. In the second phase, it continually guesses ranks from some point and verifies the guesses. Schewe proposed doing this verification in a piecemeal fashion. This yields a complement with $O((0.76n)^n)$ states, which matches the known lower bound modulo an $O(n^2)$ factor. We refer to the construction with Schewe's improvement as `Rank` in this paper.

Unlike the determinization-based approach that collects information from the history, the rank-based approach guesses ranks bounded by $2(n-|\mathcal{F}|)$ and results in many nondeterministic choices. This nondeterminism means that the rank-based construction often creates more useless states because many guesses may be verified later to be incorrect.

Slice-based approach. The slice-based construction was proposed by Kähler and Wilke in 2008 [10]. The blow-up of the construction is $4(3n)^n$ while its preliminary version in [30], referred to as `Slice` here, has a $(3n)^n$ blow-up[1]. Unlike the previous two approaches that analyze run trees, the slice-based approach analyzes reduced split trees. The construction `Slice` uses slices as states of the complement and performs a construction based on the evolution of reduced split trees in the first phase. By decorating vertices in slices at some point, it guesses whether a vertex belongs to an infinite branch of a reduced split tree or the vertex has a finite number of descendants. In the second phase, it verifies the guesses and enforces that accepting states will not occur infinitely often.

The first phase of `Slice` in general creates more states than the first phase of `Rank` because of an ordering of vertices in the reduced split trees. Similar to `Rank`, `Slice` also introduces nondeterministic choices in guessing the decorations. While `Rank` guesses ranks bounded by $2(n - |\mathcal{F}|)$ and continually guesses ranks in the second phase, `Slice` guesses only once the decorations from a fixed set of size 3 at some point.

[1] The construction in [10] has a higher complexity than its preliminary version because it treats complementation and disambiguation in a uniform way.

Assume automaton A with n states
Transition density r: there are r*n transitions for each symbol of the alphabet in A. The start and end states of these transitions are chosen at random.
r can be seen as the average number of outgoing and incoming transitions of a specific symbol per state.
Example: n = 3, r = 2: there are 6 transitions of symbol a in the automaton. As there are 3 states, each state has on average 2 outgoing transitions of a and 2 incoming transitions of a.
Acceptance density f: number between 0 and 1, there will be f*n accepting states in A. f can also be seen as the percentage of states of A that are accepting.
Example: n = 10, f = 0.4: there will be 4 accepting states in A (40% of A's states will be accepting)

4 Comparison of Complementation Approaches

We choose four representative constructions, namely `Ramsey`, `Safra-Piterman`, `Rank`, and `Slice`, that are considered the most efficient construction in each approach. These constructions are implemented in the GOAL tool [26]. We randomly generate 11,000 automata with an alphabet of size 2 and a state set of size 15 from combinations of 11 transition densities and 10 acceptance densities. For each automaton $A = (\Sigma, Q, q_0, \delta, \mathcal{F})$ with a given state size n, symbol $a \in \Sigma$, transition density r, and acceptance density f, we make $t \in \delta(s, a)$ for $\lceil rn \rceil$ pairs of states $(s, t) \in Q^2$ uniformly chosen at random and add $\lceil fn \rceil$ states to $\mathcal{F}$ uniformly at random. Our parameters were chosen to generate a large set of complementation problems, ranging from easy to hard. The experiment was run in a cluster at Rice University (http://rcsg.rice.edu/sugar/int/). For each complementation task, we allocate one 2.83 GHz CPU and 1 GB memory. The timeout of a complementation task is 10 minutes.

round up to next integer

Live states: states that occur in an accepting run of any word. All other states are dead states. Dead states can be pruned from the automaton without changing the language (p. 2, bottom).

Table 1. The results of comparing the four representative constructions

Reachable states | Live states | Percentage of live states

Constructions	Eff. Samples	S_R (Win)	S_L (Win)	S_L/S_R	T	M
`Ramsey`	-	-	-	-	11,000	0
`Safra-Piterman`	3,826	65.01 (2,797.0)	22.63 (1,066.17)	0.35	5	0
`Rank`		310.52 (1,025.5)	33.81 (1,998.67)	0.11	5,303	0
`Slice`		887.43 (3.5)	54.58 (761.17)	0.06	3,131	3,213

The 3826 automata out of 11000 that have been finished by ALL constructions.

Example for SL/SR: 100 live states and 400 (reachable) states in total: SL/SR = 100/400 = 0.25 —> 25% of all states are live states and 75% are dead states

We only collect state-size information from *effective samples*, which are tasks finished successfully by all constructions. Otherwise, a construction may be considered to be worse in producing more states because it is better in finishing more tasks. The experimental results are listed in Table 1 where S_R is the average number of reachable states created in an effective sample, S_L is the average number of live states created in an effective sample, T is the total number of timed-out tasks, and M is the total number of tasks that run out-of-memory. The Win column of S_R (resp., S_L) is the share of effective samples where one construction produces smallest complements in terms of reachable states (resp., live states). `Ramsey` is not competitive at all in complementation and is separated from the other three in Table 1 because it failed to finish any task, even though it is competitive in universality and containment, as shown in [1,5,6].

The S_R, S_L, and T columns show that the `Safra-Piterman` is the best both in average state size and in running time. The low S_L/S_R ratio shows that `Rank` and `Slice` create more dead states that can be easily pruned off. The Win columns show that although `Rank` generates more dead states, it produces more complements that are the smallest after pruning dead states. `Slice` becomes much closer to `Safra-Piterman` in the Win column of S_L because more than one half of the 3,826 effective samples are universal automata. Except `Ramsey`, `Slice` has the most unfinished tasks and produces many more states than `Safra-Piterman` and `Rank`. As we show later, we can improve the performance of `Slice` significantly by employing various optimization heuristics.

Proposed by the authors

5 Optimization Techniques

5.1 For `Safra-Piterman`

`Safra-Piterman` performs complementation via several intermediate stages: starting with the given NBW, it computes first an equivalent DPW, then the complement DPW, and finally the complement NBW. We address (1) the simplification of the complement DPW, which results in an NPW, and (2) the conversion from an NPW to an equivalent NBW.

Simplifying DPW by simulation. (`+S`). For the simplification of the complement DPW, we borrow from the ideas of Somenzi and Bloem [23]. The direct and reverse simulation relations they introduced are useful in removing transitions and possibly states of an NBW while retaining its language. We define the simulation relations for an NPW in order to apply the same simplification technique. Given an NPW $(\Sigma, Q, q_0, \delta, \mathcal{F})$ and two states $q_i, q_j \in Q$, q_j *directly simulates* q_i iff (1) for all $q_i' \in \delta(q_i, a)$, there is $q_j' \in \delta(q_j, a)$ such that q_j' directly simulates q_i', and (2) $\mathcal{F}(q_i) = \mathcal{F}(q_j)$. Similarly, q_j *reversely simulates* q_i iff (1) for all $q_i' \in \delta^{-1}(q_i, a)$, there is $q_j' \in \delta^{-1}(q_j, a)$ such that q_j' reversely simulates q_i', (2) $\mathcal{F}(q_i) = \mathcal{F}(q_j)$, and (3) $q_i = q_0$ implies $q_j = q_0$. After simplification using simulation relations, as in [23], a DPW may become nondeterministic. Therefore, the simplification by simulation can only be applied to the complement DPW.

Merging equivalent states. (`+E`). As for the conversion from an NPW to an NBW, a typical way in the literature is to go via an NRW [12,8]. We propose to go from NPW directly to NBW. Similar to the conversion from an NRW to an NBW in [12], we can nondeterministically guess the minimal even parity passed infinitely often in a run starting from some state. Once a run is guessed to pass a minimal even parity $2k$ infinitely often starting from a state s, every state t after s should have a parity greater than or equal to $2k$ and t is designated as an accepting state in the resulting NBW if it has parity $2k$. Moreover, we can make the resulting NBW smaller by merging states, with respect to an even parity $2k$, that have the same successors and have parities either all smaller than $2k$, all equal to $2k$, or all greater than $2k$. We can also start to guess the minimal even parity $2k$ starting from a state which has that parity.

5.2 For `Rank`

Maximizing Büchi acceptance set. (`+A`). As stated in Section 3, the ranks for the rank-based approach are bounded by $2(n - |\mathcal{F}|)$. The larger the $\mathcal{F}$ is, the fewer the ranks are. Thus, we propose to maximize the acceptance set of the input NBW without changing its language, states, or transition function. Given an NBW $A = (\Sigma, Q, q_0, \delta, \mathcal{F})$, we construct $A' = (\Sigma, Q, q_0, \delta, \mathcal{F}')$ with a larger acceptance set $\mathcal{F}' \supseteq \mathcal{F}$ such that $q \in \mathcal{F}'$ iff every elementary cycle containing q also contains at least one state in $\mathcal{F}$. Clearly the language of A' is the same as the language of A and we can take the complement of A' instead of A.

accepting states

This heuristic can also be applied to other complementation approaches as it maximizes the acceptance set of the input NBW before complementation. We will show the improvement made by this heuristic for `Safra-Piterman`, `Rank`, and `Slice` later in Section 6.

5.3 For `Slice`

Very similar to the construction of the lower part of the Fribourg construction

`Slice` constructs a complement with slices as states based on the evolution of a reduced split tree in the first phase, guesses the decoration for every vertex in a slice at some point, and verifies the guesses in the second phase. Intuitively, the decoration 1 indicates that a vertex must be in an infinite branch of a reduced split tree. The decoration 0 indicates that the descendants of a vertex must die out eventually before the next checkpoint. The decoration $*$ has the same meaning as 0 but the check is put on hold. In the second phase, `Slice` verifies two conditions: (1) a vertex decorated by 1 must have a right child decorated by 1, and (2) the left child of a vertex decorated by 1 and the children of a vertex decorated by 0 or $*$ must have a finite number of descendants.

Deterministic decoration. (`+D`). The first heuristic uses 1 to label vertices that *may* (rather than *must*) be in an infinite branch of a reduced split tree and only verifies the second condition in the second phase. All vertices could be decorated by 1 in the guesses. However, since the first evolution of the second phase always labels a left (accepting) child by 0 and a right (nonaccepting) child by 1, we actually decorate accepting vertices by 0 and nonaccepting vertices by 1 in the guesses. This heuristic will result in deterministic decoration. The only nondeterminism comes from choosing when to start decorating.

Reducing transitions. (`+R`). The second heuristic relies on the observation that if a run ends up in the empty sequence, a special slice denoted by $\perp$, the run will stay in $\perp$ forever and we never need to decorate the run because it can reach $\perp$ without any decoration. Thus we do not allow transitions from decorated slices other than $\perp$ to $\perp$ or from any slice to *doomed* slices; a slice is doomed if it is not $\perp$ and has no vertex labeled by 1, i.e., every run through a doomed slice is expected to reach $\perp$.

Merging adjacent vertices. (`+M`). The third heuristic recursively merges adjacent vertices decorated all by 0 or all by $*$. The observation is that they are all guessed to have a finite number of descendants and their successors will have the same decoration, either 0 or $*$.

6 Experimental Results

The heuristics proposed in Section 5 are also implemented in the GOAL tool. We use the same 11,000 automata as in Section 4 as the test bench. Since we do not propose any optimization heuristic for `Ramsey`, it is omitted in this experiment. The results showing the improvement made by the heuristics are listed in Table 2 where the Ratio columns are ratios with respect to the original construction and the other columns have the same meaning as they have in Section 4.

Compared with the original version for each construction, the experimental results in Table 2 show that (1) `Safra-Piterman+ASE` has 15 more unfinished tasks but creates almost one half of reachable states and live states, (2) the improvement made by `+A` is limited for `Safra-Piterman` and `Slice` but it is substantial for `Rank` in finishing 1,376 more tasks and avoiding the creation of around 2/3 dead states, (3) the heuristic `+D` is quite useful in reducing the reachable states down to 1/4 for `Slice` but makes more live states, and (4) `Slice+ADRM` finishes 6,116 more tasks and significantly reduces the reachable states to 1/10 and live states to one half.

Table 2. The results of comparing each construction with its improved versions

Constructions	Eff. Samples	S_R (Ratio)	S_L (Ratio)	S_L/S_R	T	M
`Safra-Piterman`	10,977	256.25 (1.00)	58.72 (1.00)	0.23	5	0
`Safra-Piterman+A`		228.40 (0.89)	54.33 (0.93)	0.24	5	0
`Safra-Piterman+S`		179.82 (0.70)	47.35 (0.81)	0.26	12	9
`Safra-Piterman+E`		194.95 (0.76)	45.47 (0.77)	0.23	11	0
`Safra-Piterman+ASE`		138.97 (0.54)	37.47 (0.64)	0.27	13	7
`Rank`	5,697	569.51 (1.00)	33.96 (1.00)	0.06	5,303	0
`Rank+A`		181.05 (0.32)	28.41 (0.84)	0.16	3,927	0
`Slice`	4,514	1,088.72 (1.00)	70.67 (1.00)	0.06	3,131	3,213
`Slice+A`		684.07 (0.63)	64.94 (0.92)	0.09	2,611	2,402
`Slice+D`		276.11 (0.25)	117.32 (1.66)	0.42	1,119	0
`Slice+R`		1,028.42 (0.94)	49.58 (0.70)	0.05	3,081	3,250
`Slice+M`		978.01 (0.90)	57.85 (0.82)	0.06	2,813	3,360
`Slice+ADRM`		102.57 (0.09)	36.11 (0.51)	0.35	228	0

Table 3. The results of comparing the three improved complementation constructions

Constructions	Eff. Samples	S_R (Win)	S_L (Win)	S_L/S_R	T	M
`Safra-Piterman+ASE`	7,045	49.94 (6,928.67)	21.38 (3,411.5)	0.43	13	7
`Rank+A`		428.61 (35.67)	41.80 (1,916.5)	0.10	3,927	0
`Slice+ADRM`		316.70 (80.67)	62.46 (1,717.0)	0.20	228	0
`Safra-Piterman+PASE`	7,593	44.84 (5,748.33)	19.50 (3,224)	0.43	4	0
`Rank+PA`		309.68 (910.33)	35.39 (2,340)	0.11	3,383	0
`Slice+PADRM`		270.68 (934.33)	53.67 (2,029)	0.20	216	0

We also compare the three constructions with all optimization heuristics in Section 5 based on 7,045 effective samples and list the results on the top of Table 3. The table shows that `Safra-Piterman+ASE` still outperforms the other two in the average state size and in running time. Table 3 also shows the following changes made by our heuristics in the comparison: (1) `Safra-Piterman+ASE` outperforms `Rank+A` in the number of smallest complements after pruning dead states, and (2) `Slice+ADRM` creates fewer reachable states than `Rank+A` in average, and finishes more tasks than `Rank+A`. As the heuristic of preminimization

applied to the input automata, denoted by `+P`, is considered to help the nondeterministic constructions more than the deterministic construction, we also compare the three constructions with preminimization and list the results in the bottom of Table 3. We only apply the preminimization implemented in the GOAL tool, namely the simplification by simulation in [23]. According to our experimental results, the preminimization does improve `Rank` and `Slice` more than `Safra-Piterman` in the complementation but does not close the gap too much between them in the comparison, though there are other preminimization techniques that we didn't implement and apply in the experiment.

7 Conclusion

We reviewed the state of Büchi complementation and examined the performance of the four complementation approaches by an experiment with a test set of 11,000 automata. We also proposed various optimization heuristics for three of the approaches and performed an experiment with the same test set to show the improvement. The experimental results show that the Safra-Piterman construction performs better than the other three in most cases in terms of time and state size. This is surprising and goes against the conventional wisdom that the nondeterministic approaches are better. The Ramsey-based construction is not competitive at all in complementation though it is competitive in universality and containment. The results also show that our heuristics substantially improve the Safra-Piterman construction and the slice-based construction in creating far fewer states. The rank-based construction and especially the slice-based construction can finish more complementation tasks with our heuristics. How the constructions scale with a growing state size, alphabet size, transition density, or other factors is not studied in this paper and is left as the future work.

References

1. Abdulla, P.A., Chen, Y.-F., Holík, L., Mayr, R., Vojnar, T.: When simulation meets antichains. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 158–174. Springer, Heidelberg (2010)
2. Althoff, C.S., Thomas, W., Wallmeier, N.: Observations on determinization of Büchi automata. Theoretical Computer Science 363(2), 224–233 (2006)
3. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proceedings of the International Congress on Logic, Method, and Philosophy of Science 1960, pp. 1–12. Stanford University Press, Stanford (1962)
4. Doyen, L., Raskin, J.-F.: Antichains for the automata-based approach to model-checking. Logical Methods in Computer Science 5(1:5), 1–20 (2009)
5. Fogarty, S., Vardi, M.Y.: Büchi complementation and size-change termination. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 16–30. Springer, Heidelberg (2009)
6. Fogarty, S., Vardi, M.Y.: Efficient Büchi universality checking. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 205–220. Springer, Heidelberg (2010)

7. Friedgut, E., Kupferman, O., Vardi, M.Y.: Büchi complementation made tighter. International Journal of Foundations of Computer Science 17(4), 851–868 (2006)
8. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
9. Gurumurthy, S., Kupferman, O., Somenzi, F., Vardi, M.Y.: On complementing nondeterministic Büchi automata. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 96–110. Springer, Heidelberg (2003)
10. Kähler, D., Wilke, T.: Complementation, disambiguation, and determinization of Büchi automata unified. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 724–735. Springer, Heidelberg (2008)
11. Karmarkar, H., Chakraborty, S.: On minimal odd rankings for Büchi complementation. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 228–243. Springer, Heidelberg (2009)
12. King, V., Kupferman, O., Vardi, M.Y.: On the complexity of parity word automata. In: Honsell, F., Miculan, M. (eds.) FOSSACS 2001. LNCS, vol. 2030, pp. 276–286. Springer, Heidelberg (2001)
13. Klarlund, N.: Progress measures for complementation of omega-automata with applications to temporal logic. In: FOCS, pp. 358–367. IEEE, Los Alamitos (1991)
14. Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of linear temporal logic. Theoretical Computer Science 363(2), 182–195 (2006)
15. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Transactions on Computational Logic 2(3), 408–429 (2001)
16. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: FOCS, pp. 531–540. IEEE Computer Society, Los Alamitos (2005)
17. Michel, M.: Complementation is more difficult with automata on infinite words. Manuscript CNET, Paris (1988)
18. Muller, D.E., Schupp, P.E.: 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)
19. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. Logical Methods in Computer Science 3(3:5), 1–21 (2007)
20. Safra, S.: On the complexity of ω-automata. In: FOCS, pp. 319–327. IEEE, Los Alamitos (1988)
21. Schewe, S.: Büchi complementation made tight. In: STACS. LIPIcs, vol. 3, pp. 661–672. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2009)
22. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with appplications to temporal logic. TCS 49, 217–237 (1987)
23. Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 248–263. Springer, Heidelberg (2000)
24. Thomas, W.: Complementation of Büchi automata revisited. In: Jewels are Forever, pp. 109–120. Springer, Heidelberg (1999)
25. Tsai, M.-H., Fogarty, S., Vardi, M.Y., Tsay, Y.-K.: State of Büchi complementation (full version), `http://goal.im.ntu.edu.tw`
26. Tsay, Y.-K., Chen, Y.-F., Tsai, M.-H., Chan, W.-C., Luo, C.-J.: GOAL extended: Towards a research tool for omega automata and temporal logic. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 346–350. Springer, Heidelberg (2008)
27. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency. LNCS, vol. 1043, pp. 238–266. Springer, Heidelberg (1996)

28. Vardi, M.Y.: Automata-theoretic model checking revisited. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 137–150. Springer, Heidelberg (2007)
29. Vardi, M.Y.: The Büchi complementation saga. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 12–22. Springer, Heidelberg (2007)
30. Vardi, M.Y., Wilke, T.: Automata: from logics to algorithms. In: Logic and Automata: History and Perspective. Texts in Logic and Games, vol. 2, pp. 629–736. Amsterdam University Press, Amsterdam (2007)
31. Yan, Q.: Lower bounds for complementation of omega-automata via the full automata technique. Logical Methods in Computer Science 4(1:5), 1–20 (2008)