Model Checking Büchi Specifications*

Deian Tabakov and Moshe Y. Vardi

Department of Computer Science, Rice University, Houston, TX
{dtabakov,vardi}@cs.rice.edu

Abstract. Efficient complementation of nondeterministic Büchi automata is essential for the automata-theoretic approach to program verification. This work presents an empirical comparison between explicit and symbolic implementations of the Kupferman-Vardi complementation construction. In order to compare the two approaches, we study automaton universality, a special case of the model checking problem. A novel encoding presented here allows the problem to be solved symbolically via a standard model checker. We compare the performance of an *explicit-search* (SPIN) and a *symbolic-search* (SMV) model checkers on randomly generated automata. We compare these results with the performance of an *explicit-encoding explicit-search* complementation tool (Wring) on the same set of automata. Our main finding is that a purely symbolic approach significantly outperforms the two other approaches.

1 Introduction

The automata-theoretic approach to program verification allows questions about the correctness of a program with respect to its specifications to be reduced to questions about language containment [29]. To check that the language of an automaton A_P is contained in the language of A_S, we check whether the intersection of A_P with the complement of A_S is empty. In the automata-theoretic framework the automaton A_P usually is an abstraction of the program and A_S represents the property that we want to verify. Thus, complementation of nondeterministic finite automata is a key problem of model checking.

If the property is given in terms of a formula φ in a temporal logic we can avoid the complementation step by first negating the formula and then constructing the corresponding automaton $A_{\neg\varphi}$ [29]. If, however, the property is given as an automaton A_S, constructing the complementary automaton $\overline{A_S}$ can be computationally demanding. Automata on finite words are not powerful enough to represent liveness and fairness properties of non-terminating programs. The framework of finite automata on infinite words provides the expressive power that we need, but we have to pay for it in terms of more expensive operations.

Automata on infinite words are usually classified according to their *acceptance condition*. Several acceptance conditions–for example, Büchi, Muller, Rabin, Streett, and parity–have been proposed and studied extensively [11]. Here we consider *Büchi automata*, where a subset of the states are designated as accepting states, and an infinite

* Supported in part by NSF grant CCR-0311326, BSF grant 9800096, and an Intel gift.

word is accepted if and only if there is a run that visits some of the accepting states infinitely often [3]. Büchi automata can be further classified according to their transition relations. In a *deterministic* automaton each state has a unique successor for each input letter. In *nondeterministic* automata a state may have multiple successors on the same input letter. Deterministic Büchi automata are strictly less expressive than nondeterministic Büchi automata [20], that is, there is a language over infinite words which is recognized by a nondeterministic Büchi automaton but cannot be recognized by any deterministic automaton. Nondeterministic Büchi automata are more natural for expressing properties of programs, but complementing those automata is more expensive. Since Büchi's original construction [3] the field has witnessed tremendous development [10, 16, 18, 21, 23].

Experimental research on Büchi complementation has been lagging behind theoretical research, which may explain why an automata-theoretic model checker such as COSPAN allows only deterministic automata to be used to specify properties [14, 19]. Tasiran et al. [28] and Althoff et al. [1] described implementations of Safra's determinization construction [23], which is the basic step for Safra's complementation. Both of these implementations had to deal with the highly involved data structures present in Safra's construction. Gurumurthy et al. [13] combined the Kupferman-Vardi construction [18] with several new minimization techniques and implemented it as an extension of the Wring tool [12]. Wring maintains the state space explicitly, and performs the complementation and minimization steps explicitly. Its performance on automata obtained from linear temporal formulas is quite good [13]. However, [13] focuses just on complementation and does not study the underlying model checking problem, nor does it evaluate performance on general Büchi automata.

There have been some attempts at a symbolic implementation in the past. Tasiran's implementation [28] uses BDDs to encode the labeled ordered trees of Safra's construction, but, as noted by the authors, the involved data structures prevents a fully symbolic implementation of the transition function. On the other hand, the simplicity of Kupferman-Vardi's construction lends itself quite naturally to a fully symbolic implementation and in this paper we present two symbolic implementations of that construction. We study complementation by looking at model checking Büchi specifications. We consider a simplified setting where the program automaton generates all infinite traces, and the model checking problem is then equivalent to checking if $\overline{A_S}$ is empty, or, equivalently, to checking if A_S is universal. This work provides the first systematic experimental evaluation of Büchi universality in the context of model checking.

Our contribution is three-fold. First, we present experimental results about the universality of nondeterministic Büchi automata generated according to the random model of [26, 31]. Under this model we vary the "hardness" of the problem by controlling the density of transitions (i.e., the ratio of transition per input letter to total states) and the density of accepting states (i.e. the ratio of accepting states to total states). We show empirically that this model does yield an interesting problem space. The probability of universality increases from 0 to 1 with the transition density and (less pronounced) with the accepting state density. We also use the random model to study the size of the complementary automaton (the tool Wring generates the complementary automaton explicitly), and show that the complementary automaton increases in size with both

transition density and accepting state density. These results confirm that this random model does provide an interesting source of benchmark problems.

Our second contribution is a direct experimental comparison between Wring and two symbolic implementations of the Kupferman-Vardi construction. To solve universality symbolically we observe that the complementary automaton can be viewed as a nondeterministic synchronous sequential circuit with fairness constraints. Checking for emptiness is equivalent to checking for the presence of a path that visits an accepting state infinitely often (a *lasso*). This can be expressed as a temporal property of the circuit, which allows us to express the universality problem as a model checking problem and solve it symbolically via a model checker. We used Cadence SMV[1] and SPIN [15]. The two model checkers present two fundamentally different ways of solving the problem. Both allow a *symbolic* encoding of the state space and the transition relation, but Cadence SMV searches for a lasso *symbolically* [5], while SPIN searches *explicitly* [7]. In contrast, Wring maintains the state space and generates the complementary automaton *explicitly*, which is then searched for a lasso using SPIN. Comparing the three approaches can give us insight into whether symbolic or explicit algorithms for Büchi complementation have better performance "in practice". Our results show that the fully symbolic approach significantly outperforms the other two. In fact, explicit construction of the complementary automaton is a challenge even for the automata with six states.

Our third contribution focuses on the key idea in the Kupferman-Vardi construction– using *ranked* subsets as the states of the complementary automata, where rank is an integer that measure "progress" toward fair termination [16]. An obvious optimization heuristic is to bound the maximum rank used in the construction. We report on empirical experiments that indicate that when attempting to establish nonuniversality, a small maximum rank is typically sufficient.

2 Preliminaries

In this section we introduce the notation used throughout this paper and review the relevant details of [18]. A *(nondeterministic) Büchi automaton* is a tuple $A = \langle \Sigma, Q, q_{in}, \rho, \alpha \rangle$, where Σ is a finite nonempty alphabet, Q is a finite nonempty set of states, $q_{in} \in Q$ is the initial state, $\alpha \subseteq Q$ is the set of accepting states, and $\rho \subseteq Q \times \Sigma \rightarrow 2^Q$ is a transition relation. A run is *accepting* iff it visits α infinitely often. A word $w \in \Sigma^\omega$ is accepted by A if A has an accepting run on w. The words accepted by A form the language of A, denoted by $L(A)$.

A *level ranking* for A is a function $g : Q \rightarrow \{0 \ldots 2n\} \cup \{\perp\}$ such that if $g(q)$ is odd, then $q \notin \alpha$. Let $\mathcal{G}$ be the set of all level rankings. For two level rankings g and g', we say that g *covers* g' if for all q and q', if $g(q) \neq \perp$ and $q' \in \delta(q, \sigma)$, then $0 \leq g'(q') \leq g(q)$ (note that when $g(q) = \perp$ the ranking of q' is left unspecified). We say that g *minimally covers* g' if g covers g' and whenever $g'(q') \neq \perp$, then there exists q such that $q' \in \delta(q, \sigma)$ and $g(q) \neq \perp$.

Let $B = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$ be a Büchi automaton. Define $N = \langle \Sigma, \mathcal{G} \times 2^Q, q'_{in}, \delta', \mathcal{G} \times \{\emptyset\} \rangle$, where

[1] `http://www.cadence.com/company/cadence_labs_research.html`

- $q'_{in} = \langle g_{in}, \emptyset \rangle$ where g_{in} is a level ranking that satisfies $g_{in}(q_{in}) = 2n$ and $g_{in}(q) = \bot$ for all $q \neq q_{in}$.
- For a state $\langle g, P \rangle \in \mathcal{G} \times 2^Q$ and a letter $\sigma \in \Sigma$, we define $\delta'(\langle g, P \rangle, \sigma)$ as follows:
  - If $P \neq \emptyset$, then $\delta'(\langle g, P \rangle, \sigma) = \{\langle g', P' \rangle : g$ minimally covers g', and $P' = \{q' :$ there is $q \in P$ such that $q' \in \delta(q, \sigma)$ and $g'(q')$ is even$\}\}$.
  - If $P = \emptyset$, then $\delta(\langle g, P \rangle, \sigma) = \{\langle g', P' \rangle : g$ minimally covers g' and $P' = \{g' : g'(q')$ is even$\}\}$.

Theorem 1 (KV01). *Let B and N be defined as above. Then $L(N) = \Sigma^\omega \backslash L(B)$.*

Intuitively, this construction is an extension of the classical subset construction for NFAs. The classical construction keeps track of subsets; here each state in the subset has an associated rank. The transitions are defined such that the ranks of the states along infinite paths are non-increasing. In addition, no state is allowed to get stuck at an even rank. To check for this we have another subset construction that keeps track of states that "owe" a pass through an odd rank (P). The acceptance condition states that infinitely often all states fulfill this obligation ($P = \emptyset$).

Recall that using the automata-theoretic approach a program P is abstracted to an automaton A_P; we want to check if it is a model of a property φ, which is represented by an automaton A_S. This is equivalent to deciding the language inclusion problem $L(A_P) \subseteq L(A_S)$ which is solved by checking $L(A_P) \cap L(\overline{A_S}) = \emptyset$. The product of the two automata has size $O(|A_P| \cdot |\overline{A_S}|)$ [6], and checking for emptiness is NLOGSPACE-complete [30]; both operations are relatively inexpensive compared to the complexity of complementing A_S. Thus, we propose to investigate the complementation part of the problem by considering a simplified setting. We set A_P to be the universal automaton, that is, $L(A_P) = \Sigma^\omega$. The model checking problem is then reduced to checking if $\overline{A_S}$ is empty, or, equivalently, whether A_S is universal. (The analogous problem for NFAs, nondeterministic automata on finite words, has received some attention recently [26, 31].

3 Experimental Setup

Random Model: In the absence of a realistic benchmark suite, we use randomly generated automata. The random model described in [26, 31] provides a framework for experimental evaluation of automata-theoretic algorithms. Here we provide a brief description; for more details see [26]. Let $A = \langle \Sigma, Q, q_{in}, \rho, \alpha \rangle$ be a Büchi automaton. In our model the alphabet Σ is the set $\{0, 1\}$. For each letter $\sigma \in \Sigma$ we generate a random directed graph D_σ on S with k edges, corresponding to transitions $\rho(q, \sigma)$. Hereafter we refer to the ratio $r = \frac{k}{|Q|}$ as the *transition density for* σ (intuitively, r represents the expected outdegree of each node for σ). We impose one exception for the initial state: when building the random graphs we make sure that the initial node has an outgoing transition for each letter of the alphabet, which helps us avoid trivial cases of non-universality. In this model the transition density of D_0 and D_1 is the same, and we refer to it as the transition density of A. The number of accepting states m is also a linear function of the total number of states, and it is given by an *acceptance density*

$f = \frac{m}{|S|}$. The accepting states themselves are selected randomly. The idea of using a linear density of some structural parameter to induce different behaviors has been quite popular lately, most notably in the context of random 3-SAT [24].

Wring: Wring [12, 25] is a tool written in Perl. Gurumurthy et al. [13] combined the Kupferman-Vardi complementation algorithm with several new minimization techniques and implemented it as an extension of Wring. Wring is a "fully explicit" tool: during complementation it maintains explicitly the set of ranked states and produces the complementary automaton explicitly. Several simulation-based minimization steps are applied at various stages of the construction, thus reducing the size of the intermediate automata[2].

While Wring produces the complementary automaton, it cannot check it for universality. For that purpose we encode the complementary automaton as a Promela model and use SPIN [15] to search for an accepting cycle. Due to the minimization steps, the automata that Wring produces are much smaller than the theoretical worst-case size. This leads to relatively small Promela models, which are model checked much faster than the time it takes to compute the complementary automaton. The times reported in our experimental results include the time spent by Wring in the complementation step, and by SPIN while it searches for an accepting cycle. Hereafter we use *Wring* to refer to the sequence of complementing an automaton using Wring and subsequently searching for a lasso using SPIN.

SPIN: SPIN [15] is an explicit-state model checker implemented in `C`. SPIN allows the specification of concurrent systems using a high-level language (`Promela`). The state-space and the transition relation can be encoded symbolically, but the state of the system is maintained explicitly. SPIN works on-the-fly, without constructing the full state-space a priori.

Cadence SMV: Cadence SMV [22] is a symbolic model checker implemented in `C`. Cadence SMV is based on *binary decision diagrams* (BDDs) [2], which provide a canonical representation for Boolean functions. BDDs are often substantially more compact than explicit representations, and have been used successfully in the verification of circuits [5].

4 Symbolic Construction

Encoding using Cadence SMV: In contrast to a recent paper by Finkbeiner [9] which describes an algorithm for language containment for Büchi automata that requires an extension to BDDs (namely, *nondeterministic BDDs*), here we present a construction that uses standard BDDs.

Let $|Q| = n$. In Cadence SMV we use two vectors, `rank` and `subset`, each of size n, such that a state (g, P) is represented by (`rank`, `subset`). The elements of `subset` are bits, and the elements of `rank` are the numbers from 0 to $2n + 1$. The value $2n + 1$ represents the element $\perp$ from the complementation construction, and the numbers $0 \ldots 2n$ represent the ranks.

[2] Two of these minimization steps, DirectSimulationMinimization() and PruneHeight(), led to incorrect answers for some automata and had to be disabled.

The initial state of the complementary automaton $\overline{A}$ is a ranking that assigns rank $2n$ to q_{in} and $\perp$ to all other states. Likewise, initially the obligation subset is empty, thus `subset` is initialized to all zeros. The transition letter is chosen nondeterministically (undefined variables can take any values from their domain).

To encode the transitions we use the `next` operator. The powerful set-constructing primitives in Cadence SMV allow us to encode very succinctly the condition that ranks are non-increasing. Finally, we assert *false* and require, using a fairness condition in the model, the obligation to be fulfilled infinitely often. In an empty model the specification holds, otherwise Cadence SMV returns a counterexample, indicating that a fair cycle has been found.

Encoding using SPIN: Encoding of the complementary automaton in Promela is based on the same ideas as the encoding in Cadence SMV. However, Promela is less expressive than Cadence SMV, making the encoding somewhat more cumbersome.

Unlike Cadence SMV, Promela does not provide a `next` operator, so here we have to maintain explicitly a vector of ranks for the current state and the next state, and likewise for the current obligation set and the next obligation set. At the beginning of each iteration a transition letter is chosen nondeterministically. For each state we select the lowest rank of its predecessors and use it to enforce that ranks are non-increasing (Appendix B). We label the location in the model where the obligation is fulfilled with `accept`, and ask SPIN to search for fair cycles.

Optimizations: A straightforward BDD-based implementation usually needs to be fine-tuned to get the best possible performance. In this section we describe optimizations that we applied to the symbolic implementation.

Fussy vs. Sloppy encoding. The Kupferman-Vardi construction can be relaxed in the following way:

- If $P \neq \emptyset$, then $\delta'(\langle g, P\rangle, \sigma) = \{\langle g', P'\rangle : g \text{ covers } g', \text{ and } P' \supseteq \{q' : \text{ there is } q \in P \text{ such that } q' \in \delta(q, \sigma) \text{ and } g'(q') \text{ is even}\}\}$.
- If $P = \emptyset$, then $\delta(\langle g, P\rangle, \sigma) = \{\langle g', P'\rangle : g \text{ covers } g' \text{ and } P' \supseteq \{g' : g'(q') \text{ is even}\}\}$.

We refer to the original encoding as *fussy*, and to the new as *sloppy*. It is easy to see why this modification does not change the universality of the automaton. Intuitively, adding more states to P' means that more states "owe" a visit to an odd rank. Similarly, in the sloppy encoding we modify the level ranking to allow states to take on an arbitrary rank instead of being assigned rank $\perp$. Notice that this optimization actually represents three different optimizations: sloppy encoding of the ranking function alone, of the obligation set alone, or both. In all three cases we are increasing the number of possible runs, but the corresponding BDDs encoding the transitions may be simpler.

Monolithic vs. Conjunctive partitioning. In [4], Burch, Clarke and Long suggest an optimization of the representation of the transition relation of a sequential circuit. They note that the transition relation is the conjunction of several small relations, and the size of the BDD representing the entire transition relation may grow as the product of the sizes of the individual parts. This encoding is called *monolithic*. The method that Burch *et al.* suggest represents the transition relation by a list of constraints, which are

implicitly conjoined. Burch *et al.* call their method *conjunctive partitioning*, which is the default encoding in Cadence SMV.

Jumping vs. Crawling. Another optimization, suggested in [18], affects how the ranks are reduced. In the original definition of level ranking, a successor state can be assigned any rank that is no larger than its predecessors'. We refer to this as *jumping*. We can restrict the level ranking such that instead of a transition to any rank smaller than i, a transition is enabled only to ranks i, $i-1$ and $i-2$ (*crawling*). This restriction does not change the language of the automaton (we can simulate one big decrease via several small ones), but it does reduce the number of possible transitions, thus it may lead to simpler BDDs.

BDD variable ordering. When using BDDs, it is crucial to select a good order of the variables. Finding an optimal order is itself a hard problem, so one has to resort to different heuristics. The default order in Cadence SMV is based on an internal heuristic. The orders that we considered included the default order, and the order given by three heuristics that are studied with respect to tree decompositions: Maximum Cardinality Search [27] (MCS), LEXP and LEXM [17]. In our experiments MCS proved to be better than LEXP and LEXM, so we will only report the results for MCS and the default order.

Forward and Backward State Traversal. Traversal of the state space can be done in two ways: we can start from the initial state and search for a strongly connected component that contains an accepting state, or start from the set of accepting states and traverse the graph backward, searching for subset that can reach itself. The advantage of *forward traversal* is that we only explore the reachable state space. On the other hand, *backward traversal* allows us to focus on the "relevant" subset of the state space. Cadence SMV allows both traversal methods to be used.

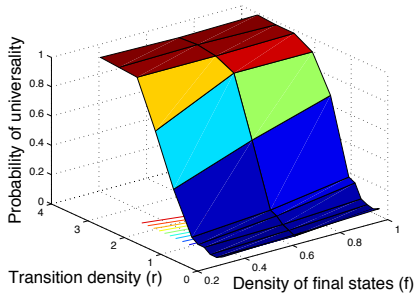
Incremental Approach. The complementation construction initially assigns rank $2|Q|$ to the initial state of the complementary automaton N and $\perp$ to every other state. However, if the source automaton B is non-universal, we might be able to find an accepting cycle starting with a lower initial ranking. Let N_i be the complementary automaton where the initial ranking assigns rank i to the initial state; the original construction then corresponds to $N_{2|Q|}$. The sequence of automata $N_0, N_1, \ldots, N_{2|Q|-1}$ under-approximates N in the sense that only a subset of the state space is explored. Consequentially, $L(N_0) \subseteq L(N_1) \subseteq \ldots \subseteq N_{2|Q|-1} \subseteq N_{2|Q|}$. If we are able to find an accepting cycle in N_i then the same cycle will be present in $N_{2|Q|}$ and both automata will accept the same word. The advantage is that the state space of N_i is much smaller than that of $N_{2|Q|}$ for small i's. We studied the minimal rank required to prove non-universality. This naturally leads to a non-universality heuristic: check non-universality with a small rank, and if no accepting cycle is found, use the full construction.

5 Experimental Results

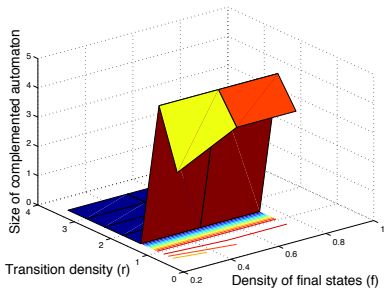
We fine tuned the performance of the symbolic approach by considering various combinations of the optimizations described in Section 4. The effect of some optimizations was independent of the rest, while others led to worse performance when used in certain combinations. We were able to achieve the best performance using conjunctive partitioning, fussy encoding of the ranking function's transitions and sloppy encoding of the

transitions of the obligation set, jumping rank reduction, and backward traversal. The results below refer to this configuration.

In Figure 1(a)[3] we present the probability of universality as a function of transition density r and accepting state density f. To generate each data point we checked the universality of 100 automata with $|Q| = 6$. The behavior that we see is not surprising. Increasing the transition density allows for more transitions between the states and more runs. Likewise, increasing the density of accepting states means that more runs are accepting.



(a) Probability of universality (automaton size 6)



(b) Average size of complemented automaton generated by *Wring* ($|Q| = 4$)

Fig. 1. The random model

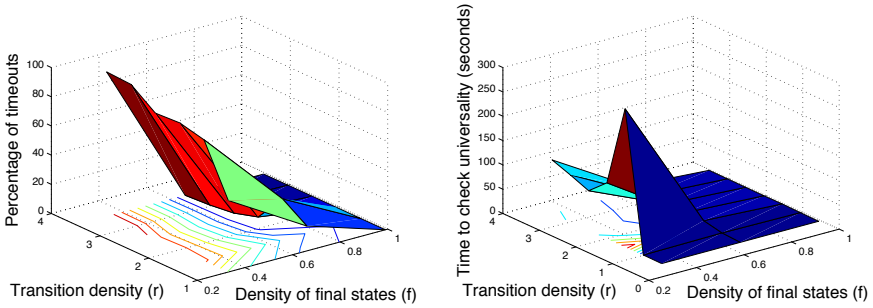
Next we turn to the size of the complementary automaton. Unlike NFAs, Büchi automata do not have a canonical normal form. Here we consider the complementary automata generated by *Wring* (Recall that *Wring* applies simulation minimization to the intermediate automata). Figure 1(b) presents the results for automata with $|Q| = 4$. Each data point represents the median size of 100 automata. The first discovery is that the median size of the complementary automaton is much smaller than the theoretical worst-case size (331,776 states). We observe that the size is biggest for small transition densities and zero for large transition densities. This behavior is quite intuitive; increasing the number of transitions means that more automata are universal, or, equivalently, that their complement is empty. *Wring* is very effective at minimizing the size of the intermediate automata and as a result, the complementary automata it produces are very small.

Landscape. In order to compare the performance of the three approaches we first use them to complement automata with a fixed size. All experiments were performed on the Rice Terascale Cluster (`http://rcsg.rice.edu/rtc/`), which is a large Linux cluster of Itanium II processors with 4 GB of memory each. For each datapoint we generated 100 random automata, checked universality using each of the methods, and then took the median. The comparison was done on random automata with $|Q| = 6$.

We first report our results for the fully explicit approach (based on *Wring*). Recall that this approach has two phases: complementing the automaton explicitly using Wring

[3] We recommend viewing the figures online in color (http://www.cs.rice.edu/ vardi/papers/).

and then searching for an accepting cycle using SPIN. Our first discovery is that the fully explicit method is unable to handle automata of that size across the whole density landscape. This is most likely due to *Wring* being written in Perl. On Figure 2(a) we show the percentage of timeouts during the complementation phase as a function of r and f (the timeout period here is 3600 seconds per automaton). The hardest problems for the fully explicit method lie in the region with low acceptance density, where the timeout range is between 60% and 100%. Everywhere else in the landscape the timeout rate is low. Our observation is that the tool either returns the complementary automaton within several hundred seconds, or times out. In all cases when Wring successfully complemented the automaton, searching for an accepting cycle took under a second. The optimizations that Wring applies to the intermediate automata lead to a very small complementary automaton, thus simplifying the job of the model checker.



(a) Timeouts with fully explicit approach (automaton size 6)

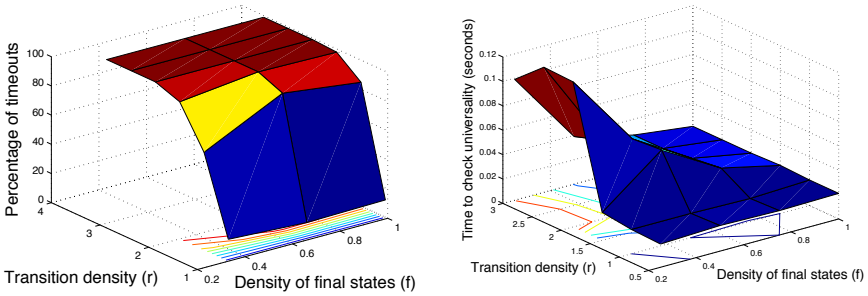
(b) Universality checking with fully explicit approach (automaton size 4)

Fig. 2. Fully explicit approach

In order to gain some insight into the time taken by the fully explicit method we made the problem simpler. Instead of using 6-state automata we used 4-state automata. Figure 2(b) presents the results. The data confirms the earlier observation that the hard problems are at low acceptance densities, however, not all of them are equally hard. Around $r = 1.5$ we observe a peak and for $r > 1.5$ the time taken is two orders of magnitude higher than for low transition densities ($r < 1,5$).

We next investigate the performance of the semi-symbolic (i.e. symbolic-encoding, explicit-search) approach using *SPIN*. As in the case for the fully explicit approach, we observe that the semi-symbolic approach times is unable to handle automata of size 6 (Figure 3(a)). We observed timeout rates of 60%-100% for $r > 1.0$ (the timeout period here is again 3600 seconds per automaton). However, if we consider automata of size 4 (Figure 3(b)) we see that the semi-symbolic approach outperforms the fully explicit approach by two and three orders of magnitude. In comparison with Figure 2(b), the peak of the graph is shifted from $r = 1.5$ to $r = 2.5$ and is less pronounced.

The data suggest that neither the fully explicit nor the semi-symbolic approach will scale for automata with more than a few states. On one hand, the fully explicit approach is very successful at minimizing the size of the intermediate automata, but the cost of the



(a) Timeouts with semi-symbolic approach (automaton size 6)

(b) Universality checking with semi-symbolic approach (automaton size 4)

Fig. 3. Semi-symbolic approach

optimizations is prohibitive when the state space is large. On the other hand, searching the full state space explicitly with the semi-symbolic approach is quite effective for small automata but infeasible when the state space increases.

Figure 4 presents the results for the fully symbolic approach using *Cadence SMV*. The immediate observation is that using the fully symbolic approach we are able to check universality for all automata with size 6 in the landscape. The hardest problems for this approach are again at low acceptance densities, with a peak at $r = 2.0$. For low transition densities ($r < 1.0$) the performance of the fully symbolic approach is worse by an order of magnitude than the semi-symbolic approach, but unlike the semi-symbolic approach, the fully symbolic one is able to handle automata with high ($r > 1.0$) transition densities.

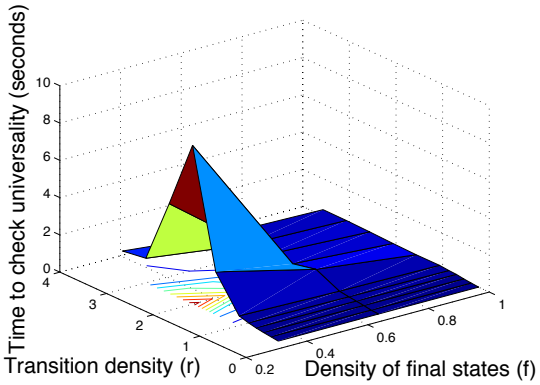


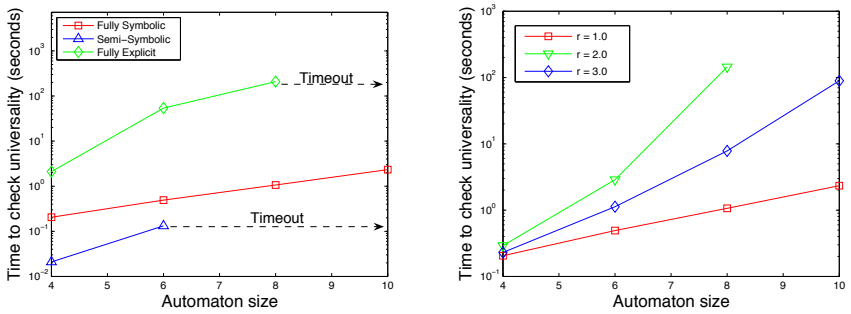
Fig. 4. Universality checking with fully symbolic approach (automaton size 6)

Our conclusion from this set of experiments is that the fully symbolic approach dominates the other two for automata with a fixed size. Next we study the scaling of the three approaches.

Scaling. A key idea from complexity theory is that performance comparison should focus on scalability, that is, how running time scales with input size. This allows us to abstract away constant factors. The key idea is that we are interested in the approach that allows us to solve the largest problems.

The results of our experiments on fixed-size automata suggest that in order to get some meaningful results we need to consider a point on the landscape where all approaches can return before the timeout period. We selected $(r, f) = (1.0, 0.5)$ and considered automata sizes between 4 and 10 in increments of 2. The data are presented on Figure 5(a). Here we see that even though the semi-symbolic approach performs the best for small automata, this does not translate into better scaling. Indeed, of all three approaches the one that scales the best is the fully symbolic, and only this approach is able to solve problems with size 10.

As pointed out before, the hardness of the problems is not uniform but rather changes as we vary the transition density. This is confirmed by our last set of experiments. In Figure 5(b) we present the scaling of universality checking with the fully symbolic method at different transition densities. As expected, the "hard" problems around $r = 2.0$ lead to the worst scaling, while problems with $r = 3.0$ and $r = 1.0$ are easier to solve even for big automata.



(a) Scaling comparison of all three approaches ($r = 1.0$, $f = 0.5$)

(b) Scaling of fully symbolic approach for different transition densities

Fig. 5. Scaling comparison

Incremental Heuristic. The last optimization described earlier (building the complementary automaton incrementally) suggests an interesting heuristic. Clearly, if the automaton is universal we will not be able to discover an accepting cycle no matter what initial ranking we choose. However, we discovered that when the starting automaton is non-universal, choosing a small initial ranking returns the result sooner, while still discovering an accepting cycle. We discovered empirically that automata with higher transition density require higher initial ranks (see Table 1). This behavior is expected: the higher the number of transitions, the harder it is to convince one that the automaton is not universal. In all cases, however, we were able to discover an accepting cycle (when one exists) using initial ranking 3 or smaller (we considered automata with sizes 6 and 8, where the original algorithm would require initial rankings of 12 and 16 accordingly). Thus, we propose the following heuristic: given an automaton, first try to

discover a cycle in the complementary automaton while using an initial ranking 3. In this case the state-space is significantly smaller than the state-space for the original algorithm, thus allowing us to get a result sooner. If we are unsuccessful we can then run the algorithm with initial ranking $2|Q|$. To test our heuristic we checked universality of automata with size 10. Using initial ranking 3 returned the correct result in 100% of the cases, and the time savings were significant. The heuristic sped up the performance by two orders of magnitude for the hardest problems ($r = 2.0$) and twice as fast for the easiest problems (at $r = 0.5$). (It is known that initial rank $2n$ is required in general [13].)

	$\|Q\| = 6$				$\|Q\| = 8$			
Transition density	rank 0	rank 1	rank 2	rank 3	rank 0	rank 1	rank 2	rank 3
r = 1.0	92	7	1	0	90	10	0	0
r = 2.0	68	32	0	0	56	36	7	1
r = 3.0	9	90	1	0	17	83	0	0

Table 1. Number of automata (out of 100 nonuniversal automata) for which an accepting cycle was discovered with the indicated initial ranking and no lower

6 Summary

We presented experimental results on the universality of Büchi automata. We showed that a fully symbolic approach scales better and is overall faster than explicit approaches. Our results indicate that the fully explicit approach is quite effective at minimizing the intermediate automata, but the time it spends on optimizing the state space is prohibitive. Searching explicitly without optimizations is also ineffective for the same reason–the state space is too large and in most cases we ran out of time. Only the fully symbolic approach was able to handle automata of reasonable size, and it scales better than the other two approaches. We also showed experimentally that in the context of the random model, using a small initial maximal rank (three) is faster up to two orders of magnitude and produces a correct answer with a very high probability (100 % in our experiments).

There are several natural extensions of this work. On one hand, in the incremental approach one can reuse the BDDs for the lower ranks as we increase the ranking of the initial state. Unfortunately, the closed-source Cadence SMV does not allow us to get access to the underlying BDDs. An open source model checker like NuSMV is an obvious choice, but our current symbolic models are not directly usable on NuSMV because NuSMV has a less expressive language. Once we have a NuSMV encoding we can add hooks into the source code of the model checker that can allow us finer control over the data structures. Further optimization of the symbolic construction presented here can be achieved by exploiting the observation in [13] that the rank of each vertex is at most $2(n - |\alpha|)$. For automata with a high number of accepting states this will lead to a significant reduction of the state space. Another direction for research is to use subsumption and keep only maximal sets of states; this idea has been used successfully

by [31] in the context of NFAs and in [8] for Büchi automata. Again, for this to be achieved we need a fine control over the underlying data structure. A third extension of this paper is to leverage the idea of *tight ranks* presented in [10]. In that work Friedgut et al. show that instead of considering all possible rankings we may restrict attention to a special class of level rankings, thus reducing the size of the reachable state space.

Our work also raises a question about the effectiveness of BDDs for this type of constructions. In the complementation construction there is a mixture of sets of numbers (the ranks) and sets of bits (the obligation sets). Encoding integers in binary representation seems unnatural, which suggests the need for hybrid data structures. We believe that this point deserves further exploration.

References

1. C. Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of Büchi automata. In Jacques Farré, Igor Litovsky, and Sylvain Schmitz, editors, *CIAA*, volume 3845 of *Lecture Notes in Computer Science*, pages 262–272. Springer, 2005.
2. R.E. Bryant. Graph-based algorithms for boolean-function manipulation. *IEEE Trans. on Computers*, C-35(8), 1986.
3. J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.
4. J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In *Proc. IFIP TC10/WG 10.5 International Conference on Very Large Scale Integration*, pages 49–58, 1991.
5. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.
6. Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and System Sciences*, 8:117–141, 1974.
7. C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.
8. Laurent Doyen and Jean-François Raskin. Improved algorithms for the automata-based approach to model-checking. In *TACAS'07, to appear*, 2007.
9. B. Finkbeiner. Symbolic refinement checking with nondeterministic BDDs. In *Tools and algorithms for the construction and analysis of systems*, LNCS. Springer-Verlag, 2001.
10. E. Friedgut, O. Kupferman, and M. Y. Vardi. Büchi complementation made tighter. In *ATVA*, pages 64–78, 2004.
11. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.
12. S. Gurumurthy, R. Bloem, and F. Somenzi. Fair simulation minimization. In *Computer Aided Verification, Proc. 14th International Conference*, volume 2404 of *LNCS*, pages 610–623. Springer-Verlag, 2002.
13. S. Gurumurthy, O. Kupferman, F. Somenzi, and M.Y. Vardi. On complementing nondeterministic Büchi automata. In *12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 2860 of *LNCS*, pages 96–110. Springer-Verlag, 2003.
14. R.H. Hardin, Z. Har'el, and R.P. Kurshan. COSPAN. In *Computer Aided Verification, Proc. 8th International Conference*, volume 1102 of *LNCS*, pages 423–427. Springer-Verlag, 1996.
15. G. J. Holzmann. *The SPIN model checker: Primer and reference manual*. Addison Wesley, 2004.

16. N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 358–367, San Juan, October 1991.
17. A. M. C. A. Koster, H. L. Bodlaender, and C. P. M. van Hoesel. Treewidth: Computational experiments. ZIB-Report 01–38, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, Germany, 2001. Also available as technical report UU-CS-2001-49 (Utrecht University) and research memorandum 02/001 (Universiteit Maastricht).
18. O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. on Computational Logic*, 2001(2):408–429, July 2001.
19. R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.
20. L.H. Landweber. Decision problems for ω–automata. *Mathematical Systems Theory*, 3:376–384, 1969.
21. C. Löding. Optimal bounds for the transformation of omega-automata. In *Proc. 19th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *LNCS*, pages 97–109, December 1999.
22. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
23. S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, White Plains, October 1988.
24. B. Selman, D. G. Mitchell, and H. J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.
25. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *CAV, Proc. 12th International Conference*, volume 1855 of *LNCS*, pages 248–263. Springer-Verlag, 2000.
26. D. Tabakov and M. Y. Vardi. Experimental evaluation of classical automata constructions. In *LPAR*, pages 396–411, 2005.
27. R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.
28. S. Tasiran, R. Hojati, and R.K. Brayton. Language containment of non-deterministic omega-automata. In *Proc. of 8th CHARME: Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 987 of *LNCS*, pages 261–277, Frankfurt, October 1995. Springer-Verlag.
29. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.
30. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.
31. M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV*, pages 17–30, 2006.

A Partial SMV encoding example

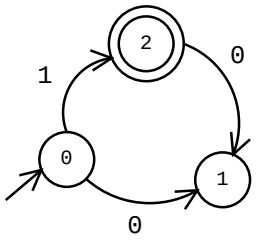


Fig. 6. Example automaton

```
module main() {
 /* The ranking function */
 rank: array 0..2 of 0..7;
 /* The P-set vector */
 subset: array 0..2 of boolean;
 /* The transition letter */
 letter: {0,1};
 /* Define the initial state. */
 init(rank) := [6,7,7];
 /* The P-set is initially empty */
 init(subset) := [0,0,0];

 /* Define the rank of state 0 in the next time step */
    next(rank[0]) := (7);
 /* Define the rank of state 1 in the next time step */
 if (letter=0) {
   next(rank[1]) := (( (rank[0] = 7) & (rank[2] = 7) ) ? 7 :
     {i: i=0..6, ((i <= rank[0]) & (i <= rank[2]) )});
 }
 else {
   next(rank[1]) := (7);
 }
 /* Defining the transitions of the P-set */
 if (subset = [0,0,0]) {
   /* The P-set is empty */
     next(subset[0]) := ( next(rank[0]) in {0,2,4,6} );
     next(subset[1]) := ( next(rank[1]) in {0,2,4,6} );
     next(subset[2]) := ( next(rank[2]) in {0,2,4,6} );
 }
 else {
   if (letter=0) {
     next(subset[2]) := 0;
```

```
   }
   else {
     next(subset[2]) := (( subset[0] ) & ( next(rank[2]) in {0,2,4,6} ));
   }
 } /* Else (P-set non empty)*/

 SPEC
   0;

 FAIRNESS
   subset = [0,0,0];

}
```

B Partial SPIN encoding example

```
bool letter[1]; /* The encoding of the transition letters */
mtype = { n7, n6, n5, n4, n3, n2, n1, n0 }

active proctype model() {

  /* The ranking function*/
  mtype  rank[3];
  mtype  next_rank[3];
  /* Encode the obligation set as a boolean vector */
  bool obligation[3] = false;
  bool next_obligation[3] = false;
  mtype min_rank;
  /* Define the initial state of the system */
  rank[0] = n6;  rank[1] = n7;  rank[2] = n7;

  do ::
       /* "Driver" for selecting the transition letter */
       if
         :: true-> letter[0] = true;
         :: true-> letter[0] = false;
       fi;
         /**     State 1   ****/
        if
           :: (letter[0] == false) ->
              /* Find the min rank of all predecessors on letter 0 */
              min_rank = n7;
              /* State 0 is a predecessor */
              if
                :: (rank[0] < min_rank) -> min_rank = rank[0];
                :: else -> skip;
              fi;
              /* State 2 is a predecessor */
              if
```

```
                  :: (rank[2] < min_rank) -> min_rank = rank[2];
                  :: else -> skip;
               fi;
               /* Make a nondeterministic choice up to min_rank */
               if
                  :: (n0 <= min_rank) -> next_rank[1] = n0;
                  :: (n1 <= min_rank) -> next_rank[1] = n1;
                      ...
                  :: (n6 <= min_rank) -> next_rank[1] = n6;
                  :: else -> next_rank[1] = n7;
               fi;
            :: else ->
               /* No predecessors for state 1 on letter 1 */
               next_rank[1] = n7;
         fi;
         ...

      /* Make the transition to the next time step */
      rank[0] = next_rank[0]; rank[1] = next_rank[1]; rank[2] = next_rank[2];

      /* Handle the obligation set */
      if
        :: (! obligation[0]) && (! obligation[1]) && (! obligation[2]) ->
               /* The obligation set is empty. Keep the even ranks */
accept:        next_obligation[0]=((next_rank[0]==n0)||...||(next_rank[0]==n6));
        :: else ->
               /* There are still obligations to be fulfilled */
               if
                  :: (letter[0] == false) ->
                     ...
                  :: else ->
                     ...
               fi; /* Transitions letter branch */
         fi; /* Empty/nonempty obligation set branch */
      /* Make the transition to the next time step */
      obligation[0] = next_obligation[0];
      ...
  od;
} /* End of model() */
```