Unfolding Büchi Automata

Jonathan Kilgallin, Klaus Sutner
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In this paper, we describe an algorithm for complementing Büchi automata. The algorithm is an optimization of Kupferman's level-ranking algorithm. The idea is to modify the definition of an automaton to allow it to read multiple characters at once. This can substantially reduce the number of states, in exchange for increasing the number of transitions. In particular, we look at using this algorithm to complement elementary cellular automata modeled by a de Bruijn graph, allowing faster model-checking for a class of first-order formulas (e.g. one universal quantifier), and provides some speed-up in the general case.

1 Introduction

Definitions: This paper assumes background working knowledge of basic automata theory, namely, NFA's and regular languages.

An ω-*automaton* is an automatic structure recognizing languages over one-way infinite words. ω-automata are analogous to NFA's, except that we have to modify the acceptance condition to account for the fact that we cannot say "The automaton accepts a word if computation on that word ends in a final state", as the computation never "ends". There are multiple equivalent conditions. The most commonly used condition is that an automaton accepts a word if computation on that word enters the set of final states infinitely often. An *omega*-automaton using this condition is called a Büchi automaton. A language recognized by some Büchi automaton is called an *omega*-regular language.

A *zeta-automaton* is the extension of an *omega*-automaton to bi-infinite words. We say that a ζ-Büchi automaton accepts a word if computation in the forward direction enters the set of final states infinitely often, and in the reverse direction enters the set of initial states infinitely often. [**?**].

A *cellular automaton* is a structure that operates on a bi-infinite array, at each time step updating the contents of each cell in the array using a local rule(for example, if the contents are zeroes and ones, we might say "at time $t+1$, put in cell i the xor of cells $i-1$, i, and $i+1$ from time t). If, in a single time step, an array configuration x evolves to a configuration y, we say that "x steps to y". An elementary cellular automaton (ECA) is one in which the local rule is always a function of the contents of the 3 local cells.

A *de Beuijn graph of rank k* is a directed graph with vertex set $\{0,1\}^k$, and edge set $\{(x,y)|$ the last $k-1$ bits of x are the first $k-1$ bits of $y\}$. This produces a graph in which every vertex has both out-degree and in-degree 2. The de Bruijn graphs up to rank 5 are drawn below.

Goal: Elementary cellular automata, while quite simple, are capable of universal computation Cite. We are interested in model-checking ECA's, considering the "steps to" relation $\rightarrow$. That is, given an ECA and a first-order predicate such as "$\exists y \forall x (x \rightarrow y)$", we wish to determine if the predicate is true of the ECA. As is shown in [**?**], this first order theory is decidable. The decision algorithm uses ζ-Büchi automata, with the automata for the basic predicate $x \rightarrow y$ built on a de Bruijn graph. Deciding an existential quantifier is simple, but in order to decide a universal quantifier, we must take the complement of an automaton (briefly, if we have an automaton for a predicate $P(x)$, to decide "$\forall x P(x)$", we take the complement, decide an existential quantifier, and complement again, using the equivalence "$\forall x P(x) = \ \exists x \ P(x)$"). Unfortunately, the complement of a ζ-Büchi automaton with n states has a worst-case lower bound of $2^{O(nlogn)}$ states cite.

Current algorithms work by first transforming the ζ-Büchi automaton into a one-way infinite Büchi automaton, and using an algorithm for complementing it. The most common such algorithm is Safra's algorithm, cite, but we will focus on a different algorithm, due to Orna Kupferman cite, and show that there is an optimization to this algorithm that greatly helps with the blow-up involved in the decision procedure.

Kupferman's Level-Ranking Algorithm: Given a Büchi automaton A, Kupferman creates an infinite directed acyclic graph (DAG) exhibiting the possible infinite paths in the automaton. This is directed graph $D = (V, E)$ using $V = Q\mathbb{N}$, with an edge from $(q1, n)$ to $(q2, n+1)$ iff there is a transition from $q1$ to $q2$. He then defines a C-ranking function $f : V \rightarrow [2n]$ satisfying that $\forall (q, l) \in V$, if $f(q, l)$ is odd, then q is not a final state, and that for every transition $q1 \rightarrow q2$, $f(q2, n+1) \leq f(q1, n)$. He defines an odd C-ranking as a C-ranking in which every path eventually remains trapped in an odd rank. He proves a lemma that every path of G has finitely many final vertices iff there is an odd C-ranking for G. He then uses the ranking functions to create another automaton to identify those paths in the graph that, after a finite number of steps, can never again reach an accepting state of the original. This algorithm produces a machine on the order of $\frac{\sqrt{6n}}{e}^n$ states, where n is the number of states in the original. expand and clean

2 Harmonic Automata

Motivation: Our improvement to Kupferman's algorithm works by modifying the definition of a Büchi automaton, allowing us to reduce the number of states. The original idea was to allow the time-dependent graph to transition more than one time-step at a time. This corresponds to allowing the original automaton to read more than one character at a time. This, in turn, suggested the idea that we select the points at which we read more than one character so that we always bypass some set of states (i.e. we pick a state q, and, whenever reading a character would put us into state q, we instead read another character in the same time step, so that we do not ever enter state q), which we can then delete. This leads to the definition of a structure we call a *Harmonic Büchi Automaton* - as reading multiple characters at a time is analogous to reading multiple notes at once in a musical score (in "harmony"), rather than a single note at a time (the "melody").

Definition: Formally, a Harmonic Automaton is a 5-tuple XXXX. The interpretation is identical to that of a Büchi automaton, with the exception that the transition function can be labeled by an arbitrarily long string, rather than a single character. Given an input word w, a state q, and an index n, the automaton may non-deterministically transition to any state from q labeled by a sequence w(n)w(n+1). This can in principle be combined with any acceptance condition, but for the remainder of the paper, we will use the following adaptation of Büchi's acceptance condition: a harmonic automata H accepts a word w if $\forall n \exists n' \geq n, q \in F(H$ can be in state q at time $n')$.

Recognized Languages: Theorem 1: The Büchi Harmonic Automata recognize the class of *omega*-regular languages. Proof: The correspondence between regular Büchi automata and harmonic Büchi automata is very simple. A Büchi automaton is already a special case of a Harmonic automaton, and the transition function can be formally modified by defining $\Delta(q, n, q')$ iff $\delta(q, w(n), q')$, where δ is the transition function for the Büchi automaton. A harmonic automaton can be made into a Büchi automaton by subdividing the transitions with "phantom" nodes, so that the transitions only read one character at a time, and visit intermediate states along the path. The insertion of these phantom nodes will henceforth be referred to as "naturalizing" the harmonic automaton. The simplicity and conciseness of this correspondence allows several of the properties of Büchi automata to be transferred directly to harmonic automata.

Unfolding: Given a Büchi automaton B, we can create a smaller, but equivalent harmonic automaton H, as long as there exists a state which satisfies the following two properties:

- It does not have a self-loop
- It is not a final node with non-final children.

Given a node that satisfies these two conditions, it can be removed, and the incoming transitions can be crossed with the outgoing transitions, to patch the transition relation. This creates a quadratic blow-up in the number of transitions, but reduces the number of states by one, but is clearly still equivalent. Iteratively removing nodes, then, can create substantially smaller machines, [examples].

Let $\mathcal{A}$ be a Büchi automaton. The *unfolding* $\mathsf{unf}(\mathcal{A})$ of $\mathcal{A}$ is the digraph on vertex set $Q\times\mathbb{N}$ with edges $(p,t) \to (q,t{+}1)$ whenever there is a transition $p \xrightarrow{s} q$ in $\mathcal{A}$ for some symbol s.

Likewise we can define the unfolding of a ζ-Büchi automaton on the vertex set $Q \times \mathbb{Z}$.

Minimizing: Theorem 2: Finding the minimal Harmonic Büchi automaton equivalent to a given Büchi automaton is NP-complete Proof: Reduction (like, it's the same problem) from feedback vertex-set, which is NP-complete [?] Upper and lower bounds for the size of a minimum harmonic automaton can be obtained from disjoint cycles and feedback vertex sets. Any set of k disjoint cycles requires at least k states in the reduced automaton, while any feedback vertex set of size k' is an upper bound on the required number of states.

Complementing: Theorem 3: The algorithm Kupferman describes still suffices for the reduced Harmonic Automata. Proof: let M be a Büchi automaton, H be its reduction, and M' be the naturalization of H. Then there is an odd level-ranking for M' iff there is one for M. Furthermore, there is an odd level-ranking for M', then there is one in which all of the phantom nodes have at every level the rank of the first non-phantom node succeeding it. Then, assigning to each node in H the rank of the corresponding node in M' gives an odd level-ranking for H. Finally, by condition 2 in the algorithm above, every path in M is α-free iff every path in H is α-free. Then, the same argument in Kupferman's paper works to give a complement for H.

The idea of a harmonic automaton is equivalent to allowing transitions of more than one level in the time-dependent graph Kupferman describes. Case analysis shows that this does not change the correctness. Given a sequence in the graph with $A \to B \to C$, any time that B is removed in the algorithm above, it does not affect the ranking for C. If A and C have opposite parity, that is, C is α-free but A is not, then there is an odd ranking function in which C's rank is A's rank minus 1. Then, depending on whether B was α-free or not, it will share either the rank of A or the rank of C. If A is α-free, then B and C must be also, so they may all be assigned the same rank. Similarly, if none of them are α-free, then they may also all be assigned the same rank.

Limitations: rewrite Unfortunately, the union and intersection constructions for harmonic automata are not immediately obvious; the standard product machine construction does not work. This means that harmonic automata lend themselves more toward complementation problems. For union and intersection, the simple solution is to transform back to a Büchi automaton, build the intersection construction, and transform it back to a harmonic automaton.

3 Application to Cellular Automata

In the special case where the Büchi automaton is a de Bruijn graph modeling a cellular automaton, this algorithm results in much smaller automata than the originals. Table 1 shows the sizes of the reduced binary de Bruijn automata up to rank 5. These values were computed by hand, and the corresponding harmonic automata for the first two entries are given below. The vertices in the final versions of the automata below form a minimum feedback vertex set in their respective original graphs. This gives a proof that these are the smallest equivalent harmonic automata. Since any reduction in the number of states is an improvement, though, and minimal automata are not necessary, one can greedily remove states for larger de Bruijn automata. We have written a program which removes nodes in numeric order. This finds the minimal automata for rank 2, 3, and 4; reduces the rank-5 automata to 9 states,and reduces the number of states by at least a factor of four for every rank between 6 and 14. Values for the size of the minimum feedback vertexset in de Bruijn graphs can be found in [**?**]. discuss removal order

In model-checking or other times when an ECA is going to be used in a product construction, then, this reduces the size of the product automaton.

size	reduced
4	3
8	4
16	6
32	8

Table 1: Sizes of reduced de Bruijn automata.

Below are examples of minimizing the first few de Bruijn graphs. Examples

We get similar performance for automata other than de Bruijn Nilpotency, Shiftcycle

In addition, we have implemented Make this statement true the adapted version of Kupferman's algorithm for Harmonic Büchi Automata, resulting in the following table:

4 Conclusion and Future Work