On Simultaneously Determinizing and Complementing
\(\omega\)-Automata\(^\dagger\)

(Extended Abstract)

E.Allen EMERSON\(^{1,2}\) and Charanjit S. JUTLA\(^1\)

1. Department of Computer Sciences, The University of Texas at Austin, USA
2. Mathematics and Computing Science Department, Technical University of Eindhoven, Netherlands

Abstract: We give a construction to simultaneously determinize and complement a Buchi Automaton on Infinite strings, with an exponential blowup in states, and linear blowup in number of pairs. An exponential lower bound is already known. The previous best construction was double exponential (Safra 88). This permits exponentially improved essentially optimal decision procedures for various Modal Logics of Programs. The new construction also gives exponentially improved conversions between various kinds of \(\omega\)-automata.

1. Introduction

Historically, automata on infinite strings were introduced by Buchi [Bu62] and slightly later by Muller [Mu63] in apparently unrelated areas: Buchi was interested in giving decision procedures for \(S1S\) (the monadic second order theory of one successor) and Muller was interested in describing behaviors of non-stabilizing asynchronous circuits. Over the years, not only have such automaton helped remove this diversity, but they now lie near the center of those areas of Computer Science where non-terminating computations are involved ([Pn77],[Pa78],[Pr79],[St81],[VW84],[VS85],[ESi84],[Sa88],[EJ88],[PR89]).

Just as in the theory of automata on finite strings, the basic theorem in automata on infinite strings relates two characterizations of languages of infinite strings: one in terms of acceptance by an automaton, the other in terms of generation by some mechanism. The second concept, that of regular \(\omega\)-event occured both in [Bu62] and [Mu63]. A language of infinite strings (or \(\omega\)-strings) \(L\) is \(\omega\)-regular iff there are regular sets \(\alpha_1...\alpha_n, \beta_1...\beta_n\) such that \(L = \alpha_1\beta_1 \cup ... \cup \alpha_n\beta_n\), where \(\alpha^\omega\) for a regular set \(\alpha\) denotes the set of all infinite strings obtained by concatenating infinitely many members of \(\alpha\).

The first notion, i.e. in terms of acceptance by finite automata, involves a complication since in contrast to automata on finite strings where acceptance can be defined in terms of a final state, automata accepting infinite strings do not reach a final state. If we allow non-determinism, then acceptance can be defined in terms of a privileged subset of states, such that an automaton accepts an \(\omega\)-string \(w\) iff the automaton non-deterministically visits some privileged state infinitely often while testing \(w\). Such an automaton is called a \textit{Buchi Automaton}. It is well known that Buchi Automata are expressively equivalent to \(\omega\)-regular languages.

As already mentioned, Buchi's motivation for studying such automata was to give a decision procedure for \(S1S\). He showed that formulae of \(S1S\) corresponded in a natural way to these automata. The decidability result required showing that such automata are closed under conjunction, disjunction and complementation. To complement a Buchi automaton, a natural way, as in the case of finite strings, seemed to be to show that Buchi Automata are equivalent to some deterministic form of automaton on infinite strings. Although, proofs without such determinization have existed ([Bu62],[SVW87]), determinization has indeed turned out to be the natural way as shown by Safra ([Sa88]).

\(^\dagger\)This work was supported in part by NSF grant DCR-8511354, ONR URI contract N00014-86-K-0763, and Netherlands NWO grant f3/nfb 62-500.
But, the real importance of this determinization process is revealed while obtaining optimal decision procedures for Branching Modal and Temporal Logics of Programs. Semantics of such logics are given with respect to infinite computation tree models. Streett [St82] showed (for one such logic, PDL-delta) that a formula of PDL-delta is satisfiable iff an automaton (effectively obtained from the formula) accepting infinite trees is non-empty. Most of these automata require checking (apart from other things) that each path of the tree belongs to some \( \omega \)-regular set. As already mentioned, it is easy to obtain a non-deterministic Buchi automaton equivalent to these \( \omega \)-regular languages. But there seems no way (at present) to obtain a tree automaton which checks that all paths are in this language, without first determinizing the Buchi automaton. If we can obtain a deterministic version of this Buchi automaton, then this automaton can be easily modified to check all paths.

A very clever notion of acceptance was given by Muller [Mu63] in his original paper, using which he defined a deterministic automaton on infinite string. McNaughton [Mc66] modified this acceptance condition (which was formalized by Rabin [Ra69] as Pairs Automaton) to prove that non-deterministic Buchi Automaton are expressively equivalent to deterministic Muller (or Rabin's Pairs) Automaton. This proof was very intricate and required that the deterministic automaton obtained from the Buchi Automaton have number of states double exponential in the number of the originial states. Safra [Sa88] gave a more transparent proof of the same theorem requiring only a single exponential state-space blowup. Safra's Construction has not only helped improve our understanding of the theory of \( \omega \)-automaton, it has also been used, along with the non-emptiness testing algorithms for tree automaton, to obtain essentially tight decision procedures for several logics of programs [EJ88] (cf. [PR89]).

If we take a closer look at the way Streett obtained the decision procedure for PDL-delta, it turns out that the tree automaton requires checking that each path of the tree belongs to the complement of some \( \omega \)-regular language. Such is also the case with most other Logics of Programs. Thus to obtain the tree automaton, it is not only required to determinize the Buchi Automaton corresponding to this \( \omega \)-regular set, but also to complement it. But, so far no single exponential blowup reduction is known which gives a deterministic Pairs automaton which accepts the complement of the original non-deterministic Buchi automaton. The reduction known so far required a double exponential blowup ([IVS85],[Sa88]). One solution of this problem was to give a new acceptance condition. This acceptance condition, known as the Complemented Pairs acceptance condition [St82], is basically the complement of the pairs condition for string automaton. For tree automaton, this acceptance condition requires that along all paths of the input tree the complement of the pairs condition hold. Given a deterministic pairs automaton on strings, the same automaton with the acceptance condition now viewed as the Complemented pairs acceptance condition, accepts the complement of the original automaton. In [EJ88], an algorithm to test non-emptiness of Complemented pairs automaton was also given, using which essentially optimal decision procedures were obtained for PDL-delta, and Mu-Calculus [Ko83].

However, there are logics for which both the determinization process, and the determinization of the complement seem to be required. A trivial example of such a logic is as follows: the atomic formulae consist of: Along all paths (and there exists a path) such that the path is in the language accepted by a non-deterministic Buchi Automaton. The formulae of the logic are boolean combinations of such atomic formulae. For such a logic, an optimal decision procedure (in this case deterministic single exponential time) uses both the determinization of a Buchi Automaton, and determinization of the complement of a Buchi Automaton in single exponential. As mentioned earlier, there was no such reduction known for the latter reduction. This problem was stated open in [Sa88].

In this paper, we build on Safra's Construction, to give a single exponential blowup construction which
simultaneously determinizes and complements a Buchi Automaton. More specifically, given a non-deterministic Buchi automaton, our construction gives a deterministic Pairs Automaton which accepts the complement of the original automaton. This is essentially an optimal conversion since a single exponential lower bound is known for complementing Buchi Automata.

The result is of particular interest for obtaining decision procedures for Modal and Temporal Logics. In particular, using the non-emptiness testing algorithm for Pairs Tree Automata in [EJ88] and our new Construction we obtain an essentially optimal non-emptiness testing algorithm for Pairs Hybrid Tree Automata [VS85]. In [VS85] for several Modal Logics of Programs, the satisfiability problem was reduced to non-emptiness problem of Hybrid Automaton, including YAPL of Vardi and Wolper [VW83], Parikh’s Game Logic [Pa83] (with the dual operator) and Streett’s Delta-Converse-PDL [St82]. It follows that YAPL has a deterministic double exponential time decision procedure (which is essentially optimal). Similarly, Propositional Game Logic (with the dual operator) and delta-converse-PDL have a deterministic single exponential time decision procedures.

Our Construction also helps us obtain essentially optimal (exponential) translations between Deterministic Complemented Pairs and Deterministic Pairs Automaton and vice-versa. The previous best known translations were again double exponential. Moreover, we can now complement a non-deterministic Complemented Pairs Automata with double-exponential blowup. The previous best known result caused triple exponential blowup.

Finally, we expect our construction to further our understanding of the theory of Automata on Infinite Objects, just as Safra’s Determinization Construct has done. The rest of the paper is organized as follows. Section 2 gives preliminary definitions and notations. In Section 3 we give our main Construction and its corollaries to the translations between different kinds of Automaton. In Section 4 we give its corollary to testing Non-emptiness of Pairs Hybrid Automata, and obtaining decision procedures for several Logics of Programs. In the concluding section 5 we discuss related work.

2. Preliminaries

2.1 \( \omega \)-Automata

An \( \omega \)-Automaton over alphabet \( \Sigma \) accepts a language which is a subset of \( \Sigma^\omega \), i.e. the set of all infinite sequences of elements from \( \Sigma \). For \( \sigma \in \Sigma^\omega \), we let \( \sigma_i \), \( i \geq 0 \), denote the \((i+1)th \) element in \( \sigma \).

An \( \omega \)-Automaton \( A \) consists of a tuple \( (\Sigma, S, \delta, s_0) \) plus an acceptance condition (described subsequently) where

- \( \Sigma \) is the input alphabet,
- \( S \) is the set of states of the automaton,
- \( \delta : S \times \Sigma \rightarrow \text{Powerset}(S) \) is the non-deterministic transition function, and
- \( s_0 \in S \) is the start state.

\( A \) is a deterministic automaton iff for each state \( s \) and each input symbol \( a : |\delta(s, a)| \leq 1 \). A run \( \rho \) of \( A \) on the input string \( \sigma \in \Sigma^\omega \) is an infinite sequence of states from \( S \), such that \( \rho_0 = s_0 \), and \( \rho_{i+1} \in \delta(\rho_i, \sigma_i) \).

We say that \( A \) accepts input string \( \sigma \) iff there exists a run \( \rho \) of \( A \) on \( \omega \) such that \( \rho \) satisfies the acceptance condition (as below). Define \( L(A) \) to be \( \{\sigma | \sigma \in \Sigma^\omega \text{ is accepted by } A \} \).

For an infinite sequence \( \rho \in \Sigma^\omega \), \( In(\rho) \) is the set \( \{i | \text{For infinitely many } i \ \rho_i = s \} \). For a Buchi Automaton ([Bu62]) \( A \) acceptance is defined in terms of a subset \( F \subseteq S \). \( \rho \) satisfies the Buchi acceptance condition iff \( In(\rho) \cap F \neq \phi \). Informally, if we traverse \( \rho \) starting from \( \rho_0 \), and flash a green light whenever we reach a state in \( F \), then \( \rho \) satisfies the Buchi acceptance condition iff the green light flashes infinitely often (for short, i.o.) on traversing \( \rho \).

For a Pairs Automaton (Rabin [Ra69]) acceptence is defined in terms of a finite list \( ((\text{RED}, \text{GREEN}), \ldots, (\text{RED}_k, \text{GREEN}_k)) \) of pairs of sets of states (think of them as pairs of colored lights where \( A \) flashes the red light of the first pair upon entering any state of the set \( \text{RED}_1 \), etc.) : \( \rho \) satisfies the pairs condition iff there exists a pair \( i \in [1..k] \)
such that REDi flashes finitely often and GREENi flashes infinitely often. More precisely, \( \exists i \in [1..k] : In(p) \cap RED_i = \phi \land In(p) \cap GREEN_i \neq \phi \). Finally, a Complemented Pairs (for Streett [St81]) automaton is defined by the above condition being false, i.e. for all pairs \( i \in [1..k] \), infinitely often GREENi flashes implies that REDi flashes infinitely often too, i.e. \( \forall i \in [1..k] : In(p) \cap GREEN_i \neq \phi \Rightarrow In(p) \cap RED_i \neq \phi \).

2.2 Hybrid Tree Automata

For notational simplicity, we only consider finite automata on infinite binary trees. An infinite binary \( \Sigma \)-tree \( T \) is a mapping \( T : (0,1)^* \rightarrow \Sigma \). A path starting at a node \( v \in (0,1)^* \) is the infinite sequence \( x = v_0, v_1, ... \) where \( v_0 = v \) and \( v_{i+1} \) is either \( v_i \cdot 0 \) or \( v_i \cdot 1 \). We let \( T[x] \) denote the infinite sequence of elements of \( \Sigma \) which occur in \( T \) along path \( x \), i.e. the infinite sequence \( T(v_0), T(v_1), ... \).

A finite automaton \( A \) on infinite binary \( \Sigma \)-trees consists of a tuple \( (\Sigma, S, \delta, s_0) \) plus an acceptance condition (described subsequently) where

- \( \Sigma \) is the input alphabet labeling the nodes of the input tree,
- \( S \) is the set of states of the automaton,
- \( \delta : S \times \Sigma \rightarrow \text{Powerset}(S^2) \) is the non-deterministic transition function, and
- \( s_0 \in S \) is the start state of the automaton.

A run of \( A \) on the input \( \Sigma \)-tree \( T \) is a function \( \rho : \{0,1\}^* \rightarrow S \) such that for all \( v \in \{0,1\}^* \), \((\rho(v0), \rho(v1)) \in \delta(\rho(v), T(v)) \) and \( \rho(\lambda) = s_0 \). We say that \( A \) accepts input tree \( T \) iff there exists a run \( \rho \) of \( A \) on \( T \) such that for all paths \( x \) starting at the root of \( T \) if \( r = \rho(x) \), the sequence of states \( A \) goes through along path \( x \), then the acceptance condition (as below) holds along \( r \).

As for Automata on infinite strings, for Buchi tree automata acceptance is defined in terms of a subset \( F \subseteq S \). \( r \) satisfies the Buchi acceptance condition iff \( \text{in}(r) \cap F \neq \phi \). For a Pairs automaton (Rabin [Ra69]) acceptance is defined in terms of a finite list \(( (RED_1, GREEN_1), \ldots, (RED_k, GREEN_k)) \) of pairs of sets of states. \( r \) satisfies the pairs condition iff there exists a pair \( i \in [1..k] \) such that \( \text{in}(r) \cap RED_i = \phi \) and \( \text{in}(r) \cap GREEN_i \neq \phi \).

A Hybrid Tree Automaton ([VS85]) \( H \) is a pair \((A, B)\), where \( A \) is a Pairs Tree Automaton and \( B \) is a Buchi \( \omega \)-Automaton, both over the same alphabet \( \Sigma \). \( H \) accepts a tree \( T \) iff \( T \) is accepted by \( A \) and, for every infinite path \( x \) starting at \( \lambda \), \( B \) does not accept (i.e. rejects) the infinite sequence \( T[x] \).

3. Technical Results

Theorem 3.1: Given a non-deterministic Buchi automata (NB, for short) \( A = (\Sigma, Q, q_0, \delta, F) \) a deterministic pairs automaton (DR, for short) \( CD = (\Sigma, Q', q_0', \delta', \Omega) \) can be constructed such that \( L(CD) \) is the complement of \( L(A) \), and \( |Q'| = 2^{|Q|^2} \) and the no. of pairs in \( \Omega \) is \( O(|Q|) \).

Introduction: Safra [Sa88] showed how to determinize a NB automata with exponential blowup in the size of the automata, and \( O(n) \) no. of pairs, where \( n \) is the size of the original NB automata. In other words, for a NB automata \( A \), Safra’s construction gives a deterministic Rabin automata (DR, for short) \( D \), such that \( \exists \text{ Run } \rho \text{ of } A \text{ on a } \omega \text{-string } s : In(p) \cap F \neq \phi \) iff \( \exists \text{ pair } i \in D : \text{ for the unique Run } \rho \text{ of } D \text{ on } s : In(p) \cap GREEN_i \neq \phi \land In(p) \cap RED_i = \phi \).

In our construction, we require \( CD \) to be such that

- \( \exists \text{ Run } \rho \text{ of } A \text{ on } s : In(p) \cap F \neq \phi \) iff
  - \( \forall \text{ Runs } \rho \text{ of } A \text{ on } s : In(p) \cap F = \phi \)

iff

- \( \exists \text{ pair } i \in CD : \text{ on the unique Run } \rho \text{ of } CD \text{ on } s : In(p) \cap GREEN_i \neq \phi \land In(p) \cap RED_i = \phi \).

The following five paragraphs require familiarity with Safra’s Construction, however, the subsequent construction and the proof are complete in themselves. The states in \( CD \) will be ordered trees as in Safra’s Construction, with the nodes labelled by subsets of \( Q \) and with two additional sets which we describe subsequently.

It is easy to see that if we imitate Safra’s Construction, by flashing a Red instead of a Green (i.e. whenever Safra’s Construction flashes a green), then an eternal node (i.e. a node which is eventually never re-
moved from the tree) in the unique run of \( CD \) flashes red infinitely often iff there exists a run \( \rho \) of \( A \) such that \( \text{In}(\rho) \cap F \neq \emptyset \).

The hard part, now, is to make sure that in case there exists a run \( \rho \) of \( A \) s.t. \( \text{In}(\rho) \cap F \neq \emptyset \), then for no other pair \( j \) in \( CD \): (infinitely often \( \text{GREEN}_j \) flashes and finitely often \( \text{RED}_j \) flashes). And if such is not the case (i.e. \( A \) rejects the input string), then there exists a pair \( j \) (i.o. \( \text{GREEN}_j \) flashes and f.o. \( \text{RED}_j \) flashes). For the second case (Completeness), by the previous paragraph it suffices to have some eternal node flash green i.o. For the first case (soundness), according to the preceding paragraph, if an eternal node \( v \) flashes red i.o. then for all other eternal nodes \( v' \), \( v \) should be blocked from flashing green i.o. unless \( v \) flashes red i.o. So, in our Construction, whenever a node \( v \) flashes red, it blocks (at least temporarily) other nodes from flashing green. But, since it is not possible to determine a priori whether \( v \) is going to flash red i.o., \( v \) itself starts flashing green until some other interesting event happens (i.e. some other node flashes red), in which case this new node takes the responsibility of flashing green.

This way Soundness is ensured, i.e. if an eternal node in \( CD \) flashes red i.o., then \( CD \) rejects. But for Completeness, the concern is that, although \( v \) on flashing red has blocked other nodes from flashing green, \( v \) may not be eternal. The naive approach of backing up using a push-down stack doesn't work.

For example, consider a node \( u1 \) which flashes red once, and never again. So, after the last red flash, it starts flashing green. And now another node \( u2 \) flashes red, and the stack becomes \( | - u1 - u2 \) where \(-\) is the bottom. \( u2 \) starts flashing green, and soon \( u3 \) flashes red, and the stack becomes \( | - u1 - u2 - u3 \). While \( u3 \) is flashing green \( u2 \) dies, and the stack has the form \( | - u1 - u3 \). Soon \( u2 \) reappears as a new node flashing red, and gets on to the top of the stack. This time, \( u3 \) dies, and reappears again, and this process continues ad infinitum. Clearly, no eternal node flashes red i.o (because \( u1 \) is the only eternal node), and hence by Safra's Construction's completeness proof there is no run in \( A \) which accepts. Hence the input should be accepted by \( CD \) (\( L(CD) \) is complement of \( L(A) \)). But, in this push-down approach no eternal node satisfies: (i.o Green and f.o Red), failing completeness of \( CD \).

In our construction, each node \( v \) that has been blocked keeps track of all nodes which flashed red between the last two times \( v \) flashed green (say \( t_{-2} \leq t < t_{-1} \)). When (if at all) all these nodes die (say at time \( t_0 \)), \( v \) flashes green. \( v \) remains blocked from flashing any more greens, except if every node (if any) which flashed red during the interval \( [t_{-1}, t_0] \) has died, in which case \( v \) is unblocked and \( v \) keeps flashing green until it is blocked again by some node flashing red.

In the actual construction, each node \( v \) will have two additional labels A-set and R-set. R-set will maintain all alive nodes which flashed red between \( [t_{-1}, t_0] \), and A-set will maintain all alive nodes which flashed red between \( [t_{-2}, t_{-1}] \).

Construction: The states of \( CD \) are labelled ordered trees as in Safra's construction. An ordered tree \( T \) is a structure \( T = (N, r, p : N \rightarrow N, S) \) where

\( N \) is a set of nodes,
\( r \) being the root node,
\( p : N \rightarrow N \) is the parenthood function defined over \( N - \{r\} \), and defining for each \( v \in N - \{r\} \), its parent \( p(v) \in N \).

\( S \) is a partial order defining "older than" on siblings (i.e. children of the same node).

For nodes \( v \) and \( v' \), if \( p(v) = v' \), then we say \( v \) is a "child of" \( v' \). "Descendant of" is the transitive closure of "child of".

A Labelled Ordered Tree is a tuple \( (T, S, A, R) \), where \( T \) is an ordered tree as above, and \( S, A, R \) are three labellings of nodes of \( T \), \( S : N \rightarrow 2^Q \), \( A : N \rightarrow 2^N \), \( R : N \rightarrow 2^N \). The label of a node \( v \) given by \( S, A, R \) will be called S-label\(_v\), A-set\(_v\), and R-set\(_v\) resp. Moreover, each node in the tree will be marked with an auxiliary color (red, green or white).

A state(-tree) of \( CD \) is a Labelled Ordered Tree in which the S-labels enjoy the following properties:

1. The union of the S-labels of the children of a
node \( v \) is a proper subset of the S-labels of \( v \).

2. The S-labels of two nodes which are not ancestral are disjoint.

The initial state \( q_0 \) is the tree of the single node labelled: S-label = Initial states of \( A \), A-set = \( \phi \), R-set = \( \phi \). The deterministic transition function \( \delta' \) transforms a state-tree, given an input \( a \in \Sigma \), by performing the following:

1. Set the color of all nodes to white.
2. For every node \( w \) with S-label \( Q' \), replace \( Q' \) by \( \delta(Q', a) \).
3. For every node \( w \) with S-label \( Q' \) s.t. \( Q' \cap F \neq \phi \), create a new node \( \tilde{w} \) which becomes the youngest son of \( v \). Mark \( \tilde{w} \) red, and set S-label, A-set, R-set = \( Q' \cap F \).
4. For every node \( w \) with S-label \( Q' \) and state \( q \in Q' \) such that \( q \) also belongs to the S-label of an older sibling \( w' \) of \( w \), remove \( q \) from \( Q' \) and all its descendants.
5. Remove all nodes with empty S-labels.
6. For every node \( w \) whose S-label is equal to the union of the S-labels of its sons, remove all the descendants of \( w \) and color \( w \) red. Moreover, set A-set, = \( \phi \), and R-set, = \( \phi \).
7. For all nodes which are removed in (5) and (6), delete these nodes from A-set and R-set of all other nodes. In other words, if \( V \subseteq A\text{-set}_w \) such that all elements of \( V \) were removed in either (5) or (6), then set A-set, = A-set, \( - V \), R-set, = R-set, \( - V \).
8. If A-set of a node \( v \) is empty and \( v \) is not marked red then color \( v \) green and set A-set, = R-set, = \( \phi \).
9. If a node is not colored red then set R-set, = R-set, \( \cup \{ n \} \) node \( n \) is colored red. \( \text{Note: } \) Nodes colored red (i.e. due to steps (3) or (6)) have R-set, and A-set, empty (see (3) and (6)).

Let \( n = |Q| \). A state \( q \in Q \) is specific to a node \( v \) if \( q \) is in the S-label of \( v \) and is not in the S-label of any other node which is not an ancestor of \( v \). It is easy to see that a state can be specific to at most one node in the state-tree. Hence, each state-tree has at most \( n \) nodes. The acceptance condition \( \Omega \) of \( CD \) will have \( n \) pairs. Whenever a new node is created in the state-tree it is assigned a pair which is not already assigned to some node in the tree. When a node is removed from the state-tree, the pair assigned to that node becomes free, and can be reassigned.

Acceptance Condition: At the end of the above round (transition) if a node is colored red, flash the red light corresponding to the pair assigned to the node.

If a node is colored green, flash the green light.

If a node is removed in (5) or (6), again flash the red light corresponding to the pair assigned to the node.

Correctness: Note that steps (1) to (6) are similar to Safra's construction, except that now in step (6) we label \( v \) red instead of green. More specifically, the creation, deletion of nodes and maintenance of S-labels of nodes in the state-tree is exactly the same.

Completeness: We show that if there is no accepting run in \( A \), i.e. no run flashes green i.o. in \( A \), then there exists a pair \( i \) in \( CD \), s.t. (i.o. Green, and f.o. Red_i).

When a node flashes green we call it the green checkpoint of the node, and when it flashes red we call it the red checkpoint.

Observation 1: If an eternal node has a last checkpoint then it must be a green checkpoint, because after the last red checkpoint (3 or 6) A-set is empty, and hence in the next round case (8) will hold.

Observation 2: Since there is no accepting run in \( A \), then if a node \( v \) is eternal, then it will not have i.m. red checkpoints. This follows from the Soundness of Safra's Construction.

Observation 3: There is at least one eternal node (root node). W.l.o.g. we can assume that all runs of \( A \) are infinite.

We will show that at least one eternal node flashes
green i.o. Since each eternal node has finitely many red checkpoints, consider the eternal node $v$ which flashed red last among the eternal nodes. After the last red checkpoint of $v$ both $A$-set$_v$ and $R$-set$_v$ become empty. And since no eternal node flashes red after this instant, no eternal node will be in the $A$-set of $v$ after this instant. And hence $A$-set of $v$ will be empty infinitely often (At each Green Checkpoint of $v$, $A$-set$_v$ is set to some finite set of nodes, all of which eventually die, making $A$-set$_v$ empty). Thus $v$ flashes green i.o. and red f.o.

**Soundness:** Suppose there is a pair which flashes green i.o. and red f.o. Then there must be an eternal node $v$ to which this pair gets assigned eventually. Let $Red_v$ be the last time $v$ flashes red (or the time of its creation, if it doesn't flash red at all).

Claim: After $Red_v$ no eternal node flashes red. First note that $A$-set of a node $v$ can become empty either because all nodes in the $A$-set die off (7) or at the red checkpoint of $v$ (6). If after $Red_v$ any eternal node $v'$ flashes red, then $v'$ will be in $R$-set$_v$ eventually. Since, $A$-set$_v$ becomes empty i.o. (because $v$ flashes green i.o.) and since there are no more red checkpoints of $v$, $v'$ will be in $A$-set$_v$ eventually (i.e. at the next green checkpoint of $v$ after $v'$ is in $R$-set$_v$). After that $A$-set$_v$ cannot become empty because $v'$ is eternal and there are no mode red checkpoints of $v$. Contradicting that $v$ flashes green i.o.

Thus, no eternal node flashes red i.o. Hence by Completeness of Safra's Construction there is no accepting run in $A$. □

For completeness sake we reprove Safra's result here. We say that a run $\psi$ at time $t$ is in node $v$ iff $\psi_t$ is in the $S$-label of $v$ at time $t$.

For completeness of Safra's Construction we prove that if there is a accepting run in $A$, then there is a node in the tree-state run of $CD$ s.t. the red light of the pair corresponding to that node flashes i.o. (note that in our construction Safra's construction is modified in the sense that it flashes red instead of flashing green). Let $\psi$ be an accepting run in $A$. Let $u$ be the (unique) deepest node (i.e. the node which has no children of that property) in the state-tree s.t. $u$ is eternal and eventually $\psi$ is always in $u$. Such a node exists because of the fact that state of $\psi$ is always in the root node which is eternal, and the depth of the state-tree is bounded. Moreover there is a unique such node because as reasoned earlier each state in $Q$ is specific to one node in the state-tree. Since $u$ is the deepest eternal node in which $\psi$ remains forever, no child of $u$ is eternal, because if some child of $u$ was eternal then some eternal child of $u$ will eventually always have $\psi$, contradicting that $u$ is the deepest.

To see the last step, note that since $\psi$ is accepting, infinitely often $u$ has a child which contains $\psi$. Then the oldest child $v$ of $u$ which ever has $\psi$ in it, will have $\psi$ in it forever (because the only way $\psi$ can be removed from a node is by (4), and by (6), each of which leads to contradiction). Thus $v$ contradicts that $u$ is the deepest. Thus $u$ flashes red i.o. (see 6, which is the only way all children of a node die).

For soundness, we prove that if an eternal node $u$ in tree-state run of $CD$ flashes red i.o., due to step 6, then there is a good (accepting) run in $A$. Let $x$ and $y$ be any two instances when $u$ flashes red due to step 6, s.t. at no instant between $x$ and $y$, $u$ flashes red due to step 6. Let the $S$-label of $u$ at $x$ and $y$ be $R_u(x)$ and $R_u(y)$. Then there must be runs in $A$ s.t. for each $s$ in $R_u(y)$ there is a run-segment, beginning at a state in $R_u(x)$ and ending at $s$, which visited a green node ($F$ node). Using these run-segments, applying Koenig's Lemma, it can be easily shown that there is an accepting run in $A$.

**Complexity:** At the end of each move of the construction, there are at most $n = |Q|$ nodes in a state tree. Therefore there are at most $2^{\land logn}$ such unlabelled trees. Even, with $S$-labels there are at most $2^{\land (n\log n)}$ state-trees. But, since each node is labelled with $A$-set and $R$-set, there can be $2^{O(n \log n)}$ labels per node, and hence the size of $CD$ is $2^{O(n \log n)}$, with $n$ pairs. □

Lemma 3.2: For a deterministic Pairs Automaton $D$
\( (\Sigma, Q, \delta, \Omega) \) with \(|Q| = n \) and \( m \) pairs in \( \Omega \), a nondeterministic Buchi automaton \( \mathcal{N} = (\Sigma, Q', \delta', F) \) can be constructed such that \(|Q'| = O(mn)\).

Proof Sketch: The construction is a standard exercise in \( \omega \)-Automaton programming. Essentially, \( \mathcal{N} \) simultaneously guesses a pair \( i \) in \( \Omega \) and a time after which \( \text{Red}_i \) never flashes. After making this guess this run of \( \mathcal{N} \) flashes Green (i.e. goes to a state in \( F \)) whenever it visits a state in \( \text{GREEN}_i \), and terminates when it visits a state in \( \text{RED}_i \).

The following Lemma is due to Vardi (see [Sa88]).

**Lemma 3.3:** For any Streett automaton with \( n \) states and \( m \) accepting pairs, an equivalent nondeterministic Buchi automaton of size \( n2^{O(m)} \) can be constructed.

**Corollary 3.4:** For any Streett automaton \( \mathcal{A} \) with \( n \) states and \( m \) accepting pairs, a deterministic Pairs automaton \( \mathcal{D} \) with \( 2^{n^2}2^{O(m)} \) states and \( n2^{O(m)} \) pairs can be constructed which accepts the complement of \( L(\mathcal{A}) \).

Proof: Apply Lemma 3.3 followed by Theorem 3.1.\( \square \)

The previous best known simultaneous determinization and complementation (in fact, even just complementation) of a Streett automaton were triple exponential. The above Theorem also gives a double exponential translation from a Non-deterministic Streett Automaton to a deterministic Streett Automaton. Double exponential translation from a Non-deterministic Streett Automaton to a deterministic Pairs Automaton is already known (Sa88)).

We next show that there is a single exponential translation from deterministic Streett to Deterministic Pairs and vice-versa. An exponential blowup lower bound can easily be shown ([Sa88]) for these two translations. The previous best known translations were double exponential ([Sa88]).

**Corollary 3.5:** For any deterministic Streett automaton (Pairs Automaton) with \( n \) states and \( m \) pairs, an equivalent deterministic Pairs Automaton (Streett Automaton resp.) with \( n2^{O(m^2)} \) states and \( O(m) \) pairs can be constructed.

Proof: Given a deterministic pairs string Automata \( \mathcal{A} \) with \( n \) states and \( m \) pairs, we can design a non-deterministic Buchi Automaton \( \mathcal{B} \) that accepts exactly those strings of states of \( \mathcal{A} \) which meet \( \mathcal{A} \)'s pairs condition. \( \mathcal{B} \) operates as follows: it guesses the pair index \( i \) certifying that the \( i \)th pair condition holds and then guesses the position along the input string of states after which no state in \( \text{Red}_i \) is ever seen again. \( \mathcal{B} \) flashes green whenever the input state is in \( \text{Green}_i \).

Thereafter, \( \mathcal{B} \) can be implemented with \( O(m) \) states. By Theorem 3.1, there is a deterministic complemented pairs Automaton \( B_1 \) with \( 2^{O(m^2)} \) states and \( O(m) \) pairs that accepts the same language as \( B \).

We now define a deterministic complemented pairs automaton \( \mathcal{A}_1 \) which is equivalent to \( \mathcal{A} \) by taking the “product” of the transition table of \( \mathcal{A} \), with \( B_1 \): on an input string over the alphabet of \( \mathcal{A} \), \( \mathcal{A}_1 \) reads the string, going through states of \( \mathcal{A} \) according to the transition table of \( \mathcal{A} \). The resulting string of states of \( \mathcal{A} \) (the run of \( \mathcal{A} \)) is fed as input (as it is generated) to \( B_1 \), whose acceptance condition determines if the run of \( \mathcal{A} \) met the pairs condition of \( \mathcal{A} \), and \( \mathcal{A}_1 \) accepts iff \( B \) accepts. Note that \( \mathcal{A}_1 \) can be implemented with \( n2^{O(m^2)} \) states and \( O(m) \) pairs.

If \( \mathcal{A} \) is a deterministic Streett Automaton, it can be viewed as a deterministic Pairs Automaton for the complement of \( L(\mathcal{A}) \). Applying the above construction to get a deterministic Streett automaton \( \mathcal{A}_1 \) for the complement of \( L(\mathcal{A}) \), and viewing \( \mathcal{A}_1 \) as a deterministic Pairs Automaton for \( L(\mathcal{A}) \), gives the conversion from deterministic Streett Automaton to deterministic Pairs Automaton.

\( \square \)

4. **Complexity of Hybrid Tree Automata**

**Theorem 4.1**[EJ88] (also see [PR89]): Non-emptiness of a pairs tree automaton with \( m \) states and \( n \) pairs can be tested in deterministic time \( O((mn)^{3n}) \).

**Corollary 4.2:** Non-emptiness of a Hybrid Automata \( \mathcal{H} = (\mathcal{A}, \mathcal{B}) \), such that \( \mathcal{A} \) has \( m_1 \) states and \( n \) pairs, and \( \mathcal{B} \) has \( m_2 \) states, can be tested in deter-
Proof: Using Theorem 3.1, obtain a deterministic Pairs string Automaton \( C \), such that \( L(C) \) is the complement of \( L(B) \), and the no. of states in \( C \) is \( 2^{O(m^2)} \) and the no. of pairs is \( m^2 \). Since \( C \) is deterministic, it can easily be converted into a Pairs tree automaton \( D \) which accepts the language: All paths of the tree are in \( L(C) \). Now, we use the standard cross-product construction to obtain a pairs Tree Automaton which accepts the language which is the intersection of \( L(A) \) and \( L(D) \). For tree automata, with \( q_1 \) and \( q_2 \) states, and \( r_1 \) and \( r_2 \) pairs resp., such a cross product construction gives a tree automaton with \( O(q_1q_22^{q_1+q_2}) \) states and \( r_1r_2 \) pairs. Thus, we obtain a tree automaton \( E \) such that \( L(E) = L(A) \cap L(D) \), and \( E \) has \( O(m1^2m2^2+n^2) \) states, and \( nm^2 \) pairs. Moreover, by definition of \( H \), \( L(E) = L(H) \). Thus, using Theorem 4.1, we can test non-emptiness of \( E \), and hence of \( H \), in deterministic time \( O(3^{m^2}n^2) \).

It is already known ([VSSS]) that the emptiness problem of Hybrid Tree Automata is logspace hard for deterministic exponential time. Thus the above algorithm is essentially optimal.

The above Theorem can be used to give essentially optimal decision procedures for several modal logics of programs. In particular, using the reduction form satisfiability of Delta-Converse-PDL to emptiness of Hybrid Automata ([Va85], [VS85]), we now obtain a deterministic exponential time algorithm for Delta-Converse-PDL. Essentially, given a formula \( f \) of Delta-Converse-PDL, a hybrid automaton \( H_f = (A, B) \) can be constructed such that \( H_f \) accepts exactly the models (with Hintikka labels) of \( f \). The automata obtained, has the property that the no. of states in \( A \) is exponential in \( |f| \), the no. of pairs in \( A \) is polynomial in \( |f| \), and the no. of states in \( B \) is polynomial in \( |f| \). Using Corollary 4.2, we get a deterministic exponential time algorithm. Similarly, satisfiability of Parikh's Game Logic ([Pa83]) can be reduced to emptiness of Tree Automata, and then using Corollary 4.2 we obtain a deterministic exponential time decision procedure for Game Logic. The above upper bounds match the known lower bounds for these logics (which is the exponential time lower bound on the decision procedure for PDL ([FL79]), which is subsumed by both these logics). The previous best known decision procedures for both these logics were of complexity non-deterministic exponential time ([Pa83] for Game Logic, and [VS85] for both the logics).

On the other hand, YAPL ([VW83]) has a deterministic double exponential lower bound ([VS85]). When we reduce satisfiability of YAPL formulae to emptiness of Hybrid Automata, the size of the string automata turns out to be exponential in the length of the original formula (in contrast to the above logics, in which the string automata are of size polynomial in the length of the original formula). This causes the decision procedure for YAPL (obtained using Corollary 4.2) to run in time deterministic double exponential time. The previous best known algorithm for YAPL was of complexity non-deterministic double exponential time ([VS85]).

5. Conclusion

We have exhibited a construction to simultaneously determinize and complement Buchi automata on infinite strings, and illustrated its application to decision procedures for Modal Logics of Programs.

Vardi [Var] has pointed out that using the reduction of satisfiability for YAPL, Game Logic, and PDL-Delta-Converse to "weak" (Buchi) Hybrid Tree Automata (in lieu of Pairs Hybrid Automata) reported in [VS85], together with results of [EJ88] and [Sa88], it is possible to obtain upper bounds for these logics similar to those above.

Finally, another single exponential determinization and complementation of Buchi string-Automata has been obtained recently by Safra [Sa].

6. Acknowledgements

We would like to thank the class of "Automata and Formal Language Theory", Fall 88, at the University of Texas at Austin, for helpful comments and suggestions. We also thank Moshe Vardi for helpful comments on the preliminary version of the paper.
References


[Var] M.Y. Vardi, personal communication


[Sa] S. Safra, "On Streett and Rabin Deterministic Automata", manuscript