A Synthesis of Two Approaches for Verifying Finite State Concurrent Systems

E. M. Clarke, Carnegie Mellon University, Pittsburgh

O. Grumberg, The Technion, Haifa

R. P. Kurshan, AT&T Bell Laboratories, Murray Hill

To be presented at

[Institution Name]

April 1981

---

1The first author was partially supported by NSF Grant MCS-82-15706. The second author was partially supported by a Weizmann postdoctoral fellowship.
A Synthesis of Two Approaches for Verifying Finite State Concurrent Systems

E. M. Clarke, Carnegie Mellon University, Pittsburgh\(^1\)

O. Grumberg, The Technion, Haifa

R. P. Kurshan, AT&T Bell Laboratories, Murray Hill

1. Introduction

Finite state concurrent systems arise in many applications. Both sequential circuits and communication protocols can be viewed as implementing such systems at some level of abstraction. When the number of system states is large, correctness may become a major problem. Two techniques have shown promise for automatically verifying this type of program. The first approach is based on temporal logic model checking and is used in the CTL verifier ([7], [8]) developed at CMU. The second approach is based on showing containment between between automata and is used by the COSPAN system developed at Bell laboratories ([1], [14]). Although the two verification systems have the same basic goal, they differ significantly in the way they attempt to achieve this goal.

The CTL model checker determines whether a formula of the propositional, branching-time logic CTL is true in some state of a labelled state-transition graph or Kripke structure. The algorithm is linear in the size of the CTL formula and also in the size of the Kripke structure. In practice, it can check state transition graphs at a rate of more than 100 states per second on a VAX 780 and has been used successfully to find subtle errors in tricky self-timed circuits ([6], [9]). Other researchers have either extended the basic CTL model checking algorithm or proposed alternative algorithms ([3], [5], [12], [15], [18], [21]).

The COSPAN system was developed at Bell Labs for protocol verification. The protocol is represented by a collection of finite state processes, \( P \). In order to show that \( P \) meets some specification \( SP \), COSPAN proves that the automaton determined by the product of the processes in \( P \) is contained in the automaton determined by \( SP \). Both the protocol and its specification are given by automata on infinite tapes in order to handle fairness properties. Usually, the protocol will be non-deterministic and its specification deterministic. The algorithm for showing containment in this case is linear in the product of the sizes of the two automata. The system has been used to verify the X.25 protocol specification, the (CCITT) FTAM protocol, and several Datakit protocols. Recently, the COSPAN system running on an IBM 3081 was able to check an automaton with more than eight million states.

Both approaches have their advantages and disadvantages. It may be difficult to describe the behaviour of a complex finite state system by a temporal logic formula. In some cases it is even impossible to do so. Automata are frequently

---

\(^1\)The first author was partially supported by NSF Grant MCS-82-16706. The second author was partially supported by a Weizmann postdoctoral fellowship
more flexible for describing such properties. Fairness properties, for example, cannot be directly expressed in CTL and must be handled indirectly by means of fairness constraints in the EMC system. Automata, on the other hand, can be tedious to debug if the number of states is large, and branching-time properties are not easily expressed in terms of language containment. Moreover, it is frequently necessary to make sure that automaton being verified satisfies specifications given by two different automata or that it satisfies one specification but not another. Thus, a logic is implicit, if not explicit, in the automata based approach.

What is needed is the ability to define new temporal operators by using automata in such a way that efficient model checking is possible. This paper provides such a synthesis: We describe a branching time temporal logic, called ECTL, which permits operators of the form $E[M(f_1, \ldots, f_n)$ and $A[M(f_1, \ldots, f_n)$ where $M$ is an automaton on infinite tapes and $f_1, \ldots, f_n$ are other ECTL formulas. (The new operators may, of course, be given more mnemonic names by the user.) Intuitively, the formula $E[M(f_1, \ldots, f_n)$ (resp. $A[M(f_1, \ldots, f_n)$) will be true in some state of a Kripke structure if some path (every path) in the structure that starts at that state is accepted by the automaton $M(f_1, \ldots, f_n)$ whose transitions are given in terms of the lower level formulas $f_1, \ldots, f_n$. All of the standard operators of branching-time temporal logic can be defined as ECTL operators. Because the operators are given by automata on infinite tapes, the logic can handle linear-time properties as well. Furthermore, we give efficient model checking algorithms for several different types of automata on infinite tapes. In each case the complexity is linear in the size of the Kripke structure and a low level polynomial in the size of the automaton $M$.

We consider, in fact, four different types of automata on infinite tapes: Büchi automata, Muller automata, L-automata, and $\forall$-automata. A Büchi automaton accepts those infinite tapes that pass through some designated set of accepting states infinitely often. The acceptance test for Muller automata involves a collection of state sets. In order for a tape to be accepted, there must be a run of the automaton on that tape such that the set of states which are visited infinitely often is one of the designated set of states. L-automata were originally proposed by Kurshan [14] and combine both Büchi and Muller acceptance conditions. $\forall$-automata were suggested as an alternative to deterministic Büchi automata in [16]; They are a like nondeterministic L-automata with the requirement that each nondeterministic path be accepting. Deterministic Muller automata and $\forall$-automata are particularly important to consider since these automata correspond to the class of $\omega$-regular languages and have not been investigated previously as the basis of an extended temporal logic.

A number of authors have proposed the use of automata on infinite tapes instead of temporal logic for verifying properties of concurrent systems ([2], [17], [16]). Although their papers argue persuasively that automata can be easier to use than temporal logic for specifying properties of concurrent programs, they do not address the problem of how the verification can be automated in the case of finite state programs. Wolper [23] has described an extension of linear temporal logic that permits operators to be specified by regular expressions. However, Sistla and Clarke [19] have shown that the model checking problem for his logic is $PSPACE$-complete. Vardi and Wolper [21] have developed an automata theoretic approach to model checking for linear temporal logic. In their approach a Büchi automaton is extracted from the formula to be checked and automata theoretic means are used to show that paths in the Kripke structure are accepted by this automaton. While their algorithm is linear in the size of the Kripke structure it is exponential in the size of the linear temporal logic formula. Since their algorithm is closely related to the decision procedure for satisfiability of linear temporal logic formulas, it would probably be difficult to implement. Vardi,
Wolper, and Sistla [24] have considered an extended version of linear temporal logic with operators specified by Büchi automata, but do not show how to handle the other types of automata that we consider or branching time properties. Vardi and Wolper [22] and Thomas [20] have proposed extended branching time logics, but have not addressed model checking problem for these logics. Habasinski [13] has investigated the model checking problem for a logic based on Muller automata, but, unfortunately, his procedure does not appear to work in general.

Our paper is organized as follows: Section 2 reviews the syntax and semantics of the computation tree logics CTL and CTL*. Section 3 describes the CTL model checker and the use of fairness constraints. Section 4 contains the definition of Muller automata and some examples of how they might be used to specify interesting properties of programs. It also contains the syntax and semantics of the logic ECTL. In the full paper this section will contain definitions and examples for Büchi automata, L-automata, and V-automata as well. Section 5 gives an efficient model checking algorithm for ECTL formulas for the case in which all of the operators are specified by Muller automata. The final paper will contain an additional section that shows how similar algorithms can be given for the other types of automata on infinite tapes. The paper concludes in Section 6 with a discussion of some open problems and directions for future research.

2. Computation Tree Logics

The logic CTL* ([8], [10], [11]) combines both branching-time and linear-time operators: a path quantifier, either A ("for all computation paths") or E ("for some computation path") can prefix an assertion composed of arbitrary combinations of the usual linear time operators G ("always"), F ("sometimes"), X ("nexttime"), and U ("until"). There are two types of formulas in CTL*: state formulas (which are true in a specific state) and path formulas (which are true along a specific path). Let AP be the set of atomic proposition names. A state formula is either:

- A, if A ∈ AP.
- If f and g are state formulas, then ¬f and f v g are state formulas.
- If f is a path formula, then E(f) is a state formula.

A path formula is either:

- A state formula.
- If f and g are path formulas, then ¬f, f v g, Xf, and f U g are path formulas.

CTL* is the set of state formulas generated by the above rules.

CTL ([4], [7]) is a restricted subset of CTL* that permits only branching-time operators—each path quantifier must be immediately followed by either an X or a U operator. More precisely, CTL is the subset of CTL* that is obtained if the syntax for path formulas is restricted to include only the following rule:

- If f and g are state formulas, then Xf and f U g are path formulas.

Linear temporal logic (LTL), on the other hand, will consist of formulas that have the form Af where f is a path formula in which the only state subformulas that are permitted are atomic propositions. More formally, a path formula is either

- An atomic proposition.
• If \( f \) and \( g \) are path formulas, then \( \neg f, f \lor g, Xf \), and \( f \cup g \) are path formulas.

We define the semantics of CTL* with respect to a structure \( K = \langle W, R, I \rangle \), where

• \( W \) is a set of states or worlds.
• \( R \subseteq W \times W \) is the transition relation, which must be total. We write \( w_i \rightarrow w_j \) to indicate that \((w_i, w_j) \in R\).
• \( L: W \rightarrow \mathcal{P}(AP) \) is a function that labels each state with a set of atomic propositions true in that state.

Unless otherwise stated, all of our results apply only to finite Kripke structures. We define a path in \( K \) to be a sequence of states, \( \pi = w_0, w_1, \ldots \) such that for every \( i \geq 0 \), \( w_i \rightarrow w_{i+1} \). \( \pi' \) will denote the suffix of \( \pi \) starting at \( w_i \).

We use the standard notation to indicate that a state formula \( f \) holds in a structure: \( K.w \models f \) means that \( f \) holds at state \( w \) in structure \( K \). Similarly, if \( f \) is a path formula, \( K,\pi \models f \) means that \( f \) holds along path \( \pi \) in structure \( K \). The relation \( \models \) is defined inductively as follows (assuming that \( f_1 \) and \( f_2 \) are state formulas and \( g_1 \) and \( g_2 \) are path formulas):

1. \( w \models A \quad \Rightarrow \quad A \in L(w) \).
2. \( w \models \neg f_1 \quad \Rightarrow \quad w \not\models f_1 \).
3. \( w \models f_1 \lor f_2 \quad \Rightarrow \quad w \models f_1 \) or \( w \models f_2 \).
4. \( w \models E(g_1) \quad \Rightarrow \quad \) there exists a path \( \pi \) starting with \( w \) such that \( \pi \models g_1 \).
5. \( \pi \models f_1 \quad \Rightarrow \quad w \) is the first state of \( \pi \) and \( w \models f_1 \).
6. \( \pi \models \neg g_1 \quad \Rightarrow \quad \pi \not\models g_1 \).
7. \( \pi \models g_1 \lor g_2 \quad \Rightarrow \quad \pi \models g_1 \) or \( \pi \models g_2 \).
8. \( \pi \models Xg_1 \quad \Rightarrow \quad \pi' \models g_1 \).
9. \( \pi \models g_1 \cup g_2 \quad \Rightarrow \quad \) there exists a \( k \geq 0 \) such that \( \pi^k \models g_1 \) and for all \( 0 \leq j < k \), \( \pi^j \not\models g_1 \).

We will also use the following abbreviations in writing CTL* (CTL and LTL) formulas:

\[
\begin{align*}
  f \land g & \equiv \neg(\neg f \lor \neg g) & \quad \text{E}f & \equiv \text{true} \lor f \\
  A(f) & \equiv \neg E(\neg f) & \quad \text{G}f & \equiv \neg F \neg f
\end{align*}
\]

In [11] it is shown that the three logics discussed in this section have different expressive powers. For example, there is no CTL formula that is equivalent to the LTL formula \( A(FGp) \). Likewise, there is no LTL formula that is equivalent to the CTL formula \( AG(\text{EF}p) \). The disjunction of these two formulas \( A(FGp) \lor AG(\text{EF}p) \) is a CTL* formula that is not expressible in either CTL or LTL.

Although \( A(FGp) \) cannot be expressed as a CTL formula, \( A(\text{GFp}) \) can be expressed as a CTL formula and in fact is equivalent to \( AG(\text{AFp}) \). We need to use this fact in Section 5. To see that it is true assume that \( A(\text{GFp}) \) is false in some state \( w \) of a Kripke structure. Hence, there must be a path \( \pi \) starting at \( w \) such that \( \neg p \) holds almost always on \( \pi \). Let \( w_1 \) be the first state on \( \pi \) such that \( \neg p \) holds at \( w_1 \) and at every state following \( w_1 \) on \( \pi \). Clearly, \( AFp \) does not hold at \( w_1 \). Since \( w_1 \) is reachable from \( w \), it must be the case that \( AG(\text{AFp}) \) is false at \( w \). Conversely, assume that \( AG(\text{AFp}) \) is false at state \( w \). Thus, \( w \models EF(\text{EG}\neg p) \). It follows that there is a state \( w \) reachable from \( w \) by a finite path.
\[ \pi, \text{ such that } w_i \models F G \neg p. \text{ There must also be a path } \pi, \text{ starting at } w_i \text{ such that } \neg p \text{ holds globally along } \pi. \text{ The path } \pi \text{ obtained by concatenating } \pi_0 \text{ and } \pi, \text{ starts at } w \text{ and shows that } \lambda(GFp) \text{ is false at that state.} \]

3. The CTL Model Checking Algorithm

Let \( K = (W, R, L) \) be a finite Kripke structure. The model checking problem for some logic \( L \) is to determine which states in \( W \) satisfy a given formula \( f \) of \( L \). This problem problem is \( PSPACE\)-complete for \( LTL \) and for \( CTL^* \). In [8], an efficient graph-traversal algorithm is given to solve model checking problem for \( CTL \).

Theorem 1: Let \( K = (W, R, L) \) be a Kripke structure and \( f \) be a \( CTL \) formula. There is an algorithm for finding the states of \( K \) where \( f \) is true that runs in time \( O(length(f) \cdot (|W| + |R|)) \).

This algorithm is implemented in the EMC system developed at CMU and has been used to debug a large number of non-trivial finite state machines ([6], [8], [9]).

Occasionally, we are only interested in the correctness of fair execution sequences. For example, we may wish to consider only execution sequences in which some process that is continuously enabled will eventually execute. This type of property cannot be expressed directly in \( CTL \) (see [11]). In order to handle such properties we must modify the semantics of \( CTL \) slightly. Initially, the model checker will prompt the user for a series of fairness constraints. Each constraint can be an arbitrary formula of the logic. A path is said to be \( fair \) with respect to a set of fairness constraints if each constraint holds infinitely often along the path. The path quantifiers in \( CTL \) formulas are now restricted to fair paths.

More formally, the new logic, which we call \( CTL^F \), has the same syntax as \( CTL \). But a structure is now a 4-tuple \( K = (W, R, L, F) \) where \( W, R, L \) have the same meaning as in the case of \( CTL \), and \( F \) is a collection of predicates on \( W, F \subseteq 2^W \). A path \( \pi \) is \( F \)-fair iff the following condition holds: for each \( G \in F \), there are infinitely many states on \( \pi \) which satisfy predicate \( G \). \( CTL^F \) has exactly the same semantics as \( CTL \) except that all path quantifiers range over fair paths. In [8] it is shown that handling fairness in this manner does not change the linear time complexity of the model checker.

Theorem 2: Let \( K = (W, R, L, F) \) be a Kripke structure with a set of fairness constraints \( F \), and let \( f \) be a \( CTL^F \) formula. There is an algorithm for finding the states of \( K \) where \( f \) is true that runs in time \( O(length(f) \cdot (|W| + |R|) \cdot |F|) \).

Fairness constraints have also been incorporated into the EMC verification system. Practical examples of their use in verifying finite state concurrent systems are given in several papers ([6], [8], [9]).

4. Automata on Infinite tapes and the logic ECTL.

A Muller Automaton is a 5-tuple \( M = (St, Alph, Tr, s_0, Freq) \) where \( St \) is a finite set of states; \( Alph \) is the input alphabet which must also be finite; \( Tr: St \times Alph \rightarrow St \) is the state transition function; \( s_0 \) is the initial state; and \( Freq \subseteq P(St) \) in a set of frequent states. An infinite string \( w = a_1a_2 \ldots \in Alph^\omega \) is accepted by \( M \) provided the set
of states that $M$ enters infinitely often when started in $s_0$ on string $w$ is one of the sets in $Freq$. More formally, let $u_0, u_1, \ldots$ be the sequence of states defined by $u_0 = s_0$ and $u_{k+1} = Tr(u_k, a_k)$. If $inf(w) = \{ s \in Sl | s = u_k$ for infinitely many $k > 0 \}$, then $w$ is accepted by $M$ iff $inf(w) \in Freq$.

In this paper $Alph$ will be the set of all possible truth assignments for some nonempty set of proposition symbols $\Sigma$. Each truth assignment will be represented by the subset of $\Sigma$ that is assigned the value true. The elements of $\Sigma$ serve as parameters when a temporal operator is defined in terms of the automaton. In describing the transitions of $M$ we will use propositional formulas over $\Sigma$ to represent subsets of $Alph$. The formula $f$ will represent all of the truth assignments in $Alph$ that satisfy $f$. For example, if $\Sigma = \{a, b\}$, then a transition from $s_1$ to $s_2$ labelled by $(a \land b) \lor (a \land \neg b)$ will actually represent two transitions from $s_1$ to $s_2$, one labelled by $\{a, b\}$ and one labelled by $\{a\}$. In using this abbreviation it is necessary to be careful that the intended automaton is really deterministic. If some state $s$ has a transition labelled with $f_1$ and another labelled with $f_2$, then it should be impossible to satisfy $f_1 \land f_2$.

Figure 4-1 shows Muller automaton $M_1$ for the until operator $U$. In this case $\Sigma = \{a, b\}$ and $Alph = \mathcal{P}\{a, b\}$. The automaton accepts infinite paths over $Alph$ that satisfy the path formula $a U b$. The set of frequent states is given by $Freq = \{\{B\}\}$. All of the other standard temporal logic operators can be defined similarly.

Figure 4-1: Muller automaton for until operator.

Figure 4-2 gives a Muller automaton $M_2$ over $\Sigma = \{a\}$ for the path formula $FGa$ (almost always $a$). Recall from Section 2 that the corresponding CTL* state formula $A(FGa)$ cannot be expressed in CTL. Although specifications involving this property frequently occur in reasoning about finite state concurrent systems, they must currently be handled by means of fairness constraints in the EMC system. The set of frequent states is given by $Freq = \{\{B\}\}$ is this case as well.

We are now ready to give the syntax and semantics of ECTL formulas. Let $\{M_i\}$ be a family of Muller automata such that each $M_i$ has $\mathcal{P}(\Sigma_i)$ as its input alphabet where $\Sigma_i = \{a_1^i, \ldots, a_k^i\}$. Let $AP$ be a set of atomic propositions. Then the set of ECTL formulas is the smallest set that is closed under the following three rules.
Figure 4-2: Muller automaton for almost always $a$.

1. Every atomic proposition is an ECTL formula.

2. If $f$ and $g$ are ECTL formulas, then $\neg f$ and $f \lor g$ are ECTL formulas.

3. If $f_1, \ldots, f_k$ are ECTL formulas, then $E(M, f_1, \ldots, f_k)$ and $A(M, f_1, \ldots, f_k)$ are ECTL formulas.

Thus, if $M_1$ and $M_2$ are the Muller automata defined in Figures 4-1 and 4-2, a typical ECTL formula might be $A(M, p \land q, q) \lor \neg E(M, q)$.

As in Section 2 we will write $K, w \models f$ to indicate that the ECTL formula $f$ is true at state $w$ in the Kripke structure $K$. The semantics of ECTL is given inductively following the syntax in the preceding paragraph. Only case 3 will be considered here since the other cases are trivial. $M(f_1, \ldots, f_k)$ will denote the Muller automaton in which each parameter $a_j^i$ of $M_i$ is replaced by the corresponding formula $f_j$. With this convention:

1. $K, w \models E(M, f_1, \ldots, f_k)$ iff for some path $\pi = w_1, w_2, \ldots$ starting at $w$ in $K$ there is an accepting sequence $C = g_1, g_2, \ldots$ of $M(f_1, \ldots, f_k)$ such that $K, w_l \models g_l$ for all $l \geq 1$.

2. $K, w \models A(M, f_1, \ldots, f_k)$ iff for every path $\pi = w_1, w_2, \ldots$ starting at $w$ in $K$ there is an accepting sequence $C = g_1, g_2, \ldots$ of $M(f_1, \ldots, f_k)$ such that $K, w_l \models g_l$ for all $l \geq 1$.

The next theorem shows that ECTL is at least as expressive as CTL$^*$. 

Theorem 3: For every CTL$^*$ formula $f$ there is an ECTL formula $f'$ which will be true in exactly the same states of a Kripke structure as $f$.

The proof of this theorem will be given in the full version of this paper.
5. Model Checking for ECTL Formulas

In order to motivate our technique we first consider two simpler problems for labelled directed graphs. Let \( G = (V, E) \) be a directed graph and let \( \text{Freq} = \{ S_1, \ldots, S_n \} \subseteq 2^V \). The E-acceptance problem for \( G \) is to find all of those vertices \( v \) such that for some path \( \pi \) starting at \( v \), \( \inf(\pi) \) is an element of \( \text{Freq} \). The A-acceptance problem for \( G \) is to find all of those vertices \( v \) such that for every path \( \pi \) starting at \( v \), \( \inf(\pi) \) is an element of \( \text{Freq} \).

We first show how to solve the E-acceptance problem by using the CTL model checker on the Kripke structure determined by \( G \) with each vertex labelled by its name. Let \( S_I = \{ v_1, \ldots, v_{n_I} \} \) be an element of \( \text{Freq} \). By a slight abuse of notation we will also use \( S_I \) to denote the propositional formula \( v_1 v_2 \ldots v_{n_I} \). We now use the CTL model checker to check the formula \( \text{EF}(\text{EG} S_I) \) with \( n_I \) fairness constraints: infinitely often \( v_1 \) infinitely often \( v_2 \) \ldots infinitely often \( v_{n_I} \). The vertices that we are interested in are the ones that are labelled with \( \text{EF}(\text{EG} S_I) \) for some \( S_I \in \text{Freq} \) when the algorithm terminates. Each such vertex is the beginning of a path \( \pi \) that visits each element of \( S_I \) infinitely often and from a certain point on is contained entirely within \( S_I \). Thus, \( \inf(\pi) = S_I \). Given the linear complexity of the CTL model checking algorithm, it is easy to see that the complexity of the E-acceptance problem is \( O(|G| \cdot |\text{Freq}|) \) where \( |G| \) is the sum of the number of vertices and the number of edges in \( G \) and \( |\text{Freq}| \) is the sum of the cardinalities of the sets in \( \text{Freq} \).

The A-acceptance problem is somewhat more complicated. First we check \( \neg \text{EG} \) true with \( n \) fairness constraints: infinitely often \( \neg v_1 \) \ldots infinitely often \( \neg v_n \). This procedure will find those vertices \( v \), such that every path starting with \( v \) is almost always within some \( S_I \). This test is not sufficient, since a path \( \pi \) might almost always be within \( S_I \) but not visit some vertex \( v_k \) of \( S_I \) infinitely often. In addition, we must insure that for every path \( \pi \) there is a set \( S_I \in \text{Freq} \) such that \( \inf(\pi) = S_I \). For each \( S_I \), we would like to check the formula

\[
A(\text{FG} S_I \rightarrow \{ \text{GF} v_1 \land \text{GF} v_2 \land \ldots \land \text{GF} v_{n_I} \})
\]

with fairness constraints: infinitely often \( \neg T_1 \) \ldots infinitely often \( \neg T_{n_I} \), where \( T_1 \ldots T_{n_I} \) are all of the elements of \( \text{Freq} \) that are subsets \( S_I \).

A vertex \( v \) that passes the test for some \( S_I \) will have the property that every path \( \pi \) which is almost always within \( S_I \), but is not almost always in any subset \( T_i \) of \( S_I \), visits every vertex in \( S_I \) infinitely often. The vertices for which the A-acceptance problem holds are those that satisfy the above formula with its fairness constraints for every \( S_I \). To see why this works, consider the case of a particular \( S_I \). Note that if a path \( \pi \) satisfies the fairness constraint infinitely often \( \neg T_j \), then \( \inf(\pi) \) is not a subset of \( T_j \). Thus, if \( \pi \) is fair, \( \inf(\pi) \) is not contained in any subset \( T_j \ldots T_{n_I} \) of \( S_I \). Hence, if \( \pi \) is almost always contained within \( S_I \), then it should visit each of the elements of \( S_I \) infinitely often. If \( \pi \) is not fair, then \( \inf(\pi) \) is a subset of some \( T_i \) and its acceptance will be determined when \( T_i \) is considered.

Unfortunately, the above formula is not a CTL formula and, therefore, cannot be directly checked using the CTL model checker. However, we can rewrite it as

\[
A(\text{FG} S_I \rightarrow \text{GF} v_1) \land A(\text{FG} S_I \rightarrow \text{GF} v_2) \land \ldots \land A(\text{FG} S_I \rightarrow \text{GF} v_{n_I})
\]

Thus, it is sufficient to be able to check formulas of the form \( A(\text{FG} p \rightarrow \text{GF} q) \) with a series of fairness constraints:
infinitely often \( u_i \), \ldots, infinitely often \( u_d \). This formula is still not a CTL formula, but it is equivalent to one as the following chain of identities shows.

\[
\begin{align*}
\Lambda(FG \, p \rightarrow GF \, q) & \equiv \Lambda(\neg FG \, p \lor GF \, q) \\
& \equiv \Lambda((GF \, \neg p) \lor GF \, q) \\
& \equiv \Lambda(GF \, (\neg p \lor q)) \\
& \equiv \Lambda(AF \, (\neg p \lor q))
\end{align*}
\]

The last equivalence follows from the identity \( \Lambda(GF \, f) \equiv \Lambda(AF \, f) \) discussed in Section 2. The last formula can be checked by the standard model checking algorithm. It follows that the complexity of the \( A \)-acceptance problem is \( O(|G| \cdot |Freq|^3) \) where \( |G| \) and \( |Freq| \) are as above.

We would like to use the technique described above to check temporal operators defined in terms of Muller automata. In order to accomplish this we first define a new Kripke structure which is the product of the Kripke structure to be checked and the given Muller automaton. Let \( K = (W, R, L) \) be the Kripke structure and \( M = (St, \, Alph, \, Tr, \, s_0, \, Freq) \) be a complete Muller automaton, then \( K \times M \) is the Kripke structure with state set \( W \times St \) such that each state \( (w, s) \in W \times St \) is labelled by \( \{s\} \). The transition relation for the product Kripke structure is given by the following rule: There will be a transition \( (w, s) \rightarrow (w', s') \) provided that

1. \( w \rightarrow w' \) is a transition of \( K \).
2. \( Tr(s, g) = s' \) is a transition of \( M \).
3. \( w \models g \) holds in \( K \).

A path \( p \) in the product structure \( K \times M \) can be decomposed into a path \( \pi \) in \( K \) and a path \( c \) in \( M \). The path \( p \) simulates the behavior of \( M \) on \( \pi \). More precisely, if \( c \) starts at the initial state of \( M \) and \( inf(c) \) is an element of \( Freq \), then path \( \pi \) in \( K \) is accepted by the Muller automaton \( M \). The completeness of \( M \) is a technical restriction which insures that the transition relation of \( K \times M \) is total, i.e. every state in the product structure has at least one successor. A Muller automaton that is not complete can be converted to one that is by adding "trap" states. A somewhat more complicated definition of the product structure can also be used to avoid this assumption.

We now show how to check \( E[M](f_1, \ldots, f_m) \) and \( A[M](f_1, \ldots, f_m) \) assuming that \( f_1, \ldots, f_m \) are atomic formulas labelling \( K \).

**Theorem 4:** \( K, w \models E[M](f_1, \ldots, f_m) \) iff \( K \times M(f_1, \ldots, f_m), (w, s_0) \models EF(Egf \, S) \) with fairness constraints: infinitely often \( s_1, \ldots, s_{i_1} \), infinitely often \( s_{l_1}, \ldots, s_{l_1} \), for some \( S_t = \{s_1, \ldots, s_{l_1}\} \in Freq \).

The complexity for this case is \( O(|K| \cdot |Tr| \cdot |Freq|) \), where \( |Tr| \) is the size of \( M \)'s transition graph and \( |Freq| \) is the size of its frequent set measured as in the directed graph case considered previously.

**Theorem 5:** \( K, w \models A[M](f_1, \ldots, f_m) \) iff
1. $\text{K} \times \text{M}(f_1, \ldots, f_m), (w, s_0) \models \neg \text{FG} \text{true with fairness constraints: infinitely often } \neg S_1, \ldots$
   \hspace{1cm} infinitely often $\neg S_n$.

2. For every $S_i \in \text{Freq}$:
   
   $$\text{K} \times \text{M}(f_1, \ldots, f_m), (w, s_0) \models \Lambda(\text{FG } S_i \rightarrow [\text{GF } s_1 \land \text{GF } s_2 \land \ldots \land \text{GF } s_\eta])$$

   with fairness constraints: infinitely often $\neg T_1, \ldots$, infinitely often $\neg T_\eta$, where $T_1, \ldots, T_\eta$ are all of the elements of $\text{Freq}$ that are subsets $S_i$.

   The complexity for this case is $O(|K| \cdot |\text{Tr}| \cdot |\text{Freq}|^2)$.

The algorithm outlined above will label each state $(w, s_0)$ of $\text{K} \times \text{M}$ with an $\Lambda$ or an $E$ formula iff that formula is true in state $w$ of $K$. We can use the labelling of the product graph to label the states of the original Kripke structure appropriately. In order to handle an arbitrary ECTL formula $h$, we successively apply the state labeling algorithm to the subformulas of $h$, starting with the shortest, most deeply nested and work outward to include all of $h$. Since processing a single Muller automaton $M_i$ with $\text{Tr}_i$ as its transition relation and $\text{Freq}_i$ as its acceptance set takes time $O(|K| \cdot |\text{Tr}_i| \cdot |\text{Freq}_i|^2)$ and since $h$ has $\text{length}(h)$ different subformulas, the entire algorithm requires $O(|K| \cdot \text{length}(h) \cdot |\text{Tr}_{\text{max}}| \cdot |\text{Freq}_{\text{max}}|^2)$, where $\text{max}$ is the index of the Muller automaton used as an operator in $h$ for which the product $|\text{Tr}_i| \cdot |\text{Freq}_i|^2$ is largest. Note that this complexity is linear in the size of the Kripke structure $K$ and a low order polynomial in the size of $M_{\text{max}}$.

6. Conclusion

It should be quite easy to adapt the EMC system to handle ECTL formulas. Since the basic algorithms in this paper are given in terms of primitives that are already implemented in the EMC verifier, it should only be necessary to provide a parser for ECTL formulas and a program for computing the product of a Kripke structure and the appropriate automaton on infinite tapes. Since memory is always a critical resource with this type of verification, a version of the algorithm that avoids the product construction would be quite useful. Such an algorithm has been obtained for the case of Büchi automata and L-automata by T.G. Tang at CMU. Although his algorithm will use as much memory in the worst case as the algorithms described here, it is possible to give examples where it results in enormous savings.

An obvious question to ask is whether our techniques can be extended to handle nondeterministic versions of the automata discussed in this paper. It is not difficult to show that the model checking problem is $\text{PSPACE-hard}$ for ECTL formulas with operators specified by nondeterministic Büchi automata. Although an algorithm for this may still be polynomial in the size of the Kripke structure, it will likely involve some version of the closure construction in ([21], [24]) and will probably be much more difficult to implement. Similar results should hold for the other types of automata as well. Other theoretical questions require more research. For example, there is a close relationship between the model checking problem and various problems for formal languages like checking emptiness and containment of languages. It would be interesting to investigate this relationship further. In particular, the algorithm
described in Section 5 should give a simple polynomial algorithm for deciding containment of deterministic Muller automata. Another problem to consider is the complexity of deciding validity for ECTL formulas. Although the ECTL has provably greater expressive power than CTL*, the model checking problem is of much lower complexity for ECTL than for CTL*. Is this lower complexity observed for validity as well?

Finally, since this paper has described how several different types of automata can be used for model checking, it is natural to ask which is best? The translation from a deterministic Büchi automaton to a deterministic Muller automata is exponential in general. The reverse translation (from deterministic Muller to deterministic Büchi) is not always possible. Thus, it is unlikely that any one type of automaton is best for all problems. Some problems can be specified more succinctly using Büchi automata and some using Muller automata. L-automata and V-automata are a step in the right direction since they combine Büchi and Muller acceptance properties. Our approach provides for maximum flexibility since the CTL model checking algorithm can be used as the basis for efficient checking algorithms for all four types of automata.

Acknowledgment: We wish to acknowledge the help of T. G. Tang.

References


