A Synthesis of Two Approaches for Verifying Finite State Concurrent Systems

E. M. CLARKE, *Carnegie Mellon University, Pittsburgh, USA*

O. GRUMBERG, *Technion-Israel Institute of Technology, Haifa, Israel*

R. P. KURSHAN, *AT&T Bell Laboratories, 600 Mountain Ave, Murray Hill, NJ 07974, USA*

Abstract

The paper provides a synthesis between two main approaches to automatic verification of finite-state systems: temporal logic model checking and language containment of automata on infinite tapes. A new branching-time temporal logic is suggested, in which automata on infinite tapes are used to define new temporal operators. Each such operator defines a set of acceptable computation paths. Path quantifiers are used to specify whether *all* paths or *some* path from a state should be in some acceptable set. The logic is very powerful and includes both linear-time and branching-time temporal logics. We give an efficient model checking procedure that checks whether a finite-state system satisfies its specification, given by a formula of the new logic. Our procedure is linear in the size of the system and a low level polynomial in the size of the specification.

Keywords: Automata on infinite tapes, finite-state systems, model checking, temporal logics, ω-regular languages.

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 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 basic algorithm is linear in the size of the CTL formula and also in the size of the Kripke structure. It has been used successfully to find subtle errors in self-timed circuits [5, 9]. A number of other researchers have either extended

the basic algorithm or proposed alternative algorithms [3, 6, 12, 15, 20, 23]. Recently, a new version of the CTL model checking algorithm has been developed which uses binary decision diagrams [4] to represent state-transition graphs in a very concise manner. The new algorithm has permitted much larger state-spaces to be searched than was previously possible. For example, in [17] it was used to verify certain properties of the cache consistency protocol for a new multiprocessor being developed by the Encore Corporation (the Encore Gigamax). This required searching a state-space with more than 10^{13} states.

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. An implementation of the COSPAN system based on binary decision diagrams is currently being developed.

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 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 CTL verifier. 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 the automaton being verified satisfies specifications given by two different automata or that is 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 $\mathbf{E}[M](f_1, \ldots, f_n)$ and $\mathbf{A}[M](f_1, \ldots, f_n)$ 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 $\mathbf{E}[M](f_1, \ldots, f_n)$ (resp. $\mathbf{A}[M](f_1, \ldots, f_n)$) 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. This approach is applicable to a number of 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.

A number of authors have proposed the use of automata on infinite tapes instead of temporal logic for verifying properties of concurrent systems [2, 16, 19]. 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 [25] has described an extension of linear temporal logic that permits operators to be specified by regular expressions. However, Sistla and Clarke [21] have shown that the model checking problem for his logic is PSPACE-complete. Vardi and Wolper [23] 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 closely related to the decision procedure for satisfiability of linear temporal logic formulas, it would probably be difficult to implement. Vardi, Wolper, and Sistla [26] have considered an extended version of linear temporal logic with operators specified by Büchi automata, but do not show how to handle the branching time properties. Vardi and Wolper [24] and Thomas [22] 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 Müller 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 Müller 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. Section 5 gives an efficient model checking algorithm for ECTL formulas for the case in which all the operators are specified by Müller automata. 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 $A \in AP$.
- If f and g are state formulas, then $\neg f$ and $f \vee g$ are state formulas.
- If f is a path formula, then $\mathbf{E}(f)$ is a state formula.

A path formula is either:

- A state formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $\mathbf{X}f$, and $f\mathbf{U}g$ are path formulas.

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

CTL 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 $\mathbf{X}f$ and $f\mathbf{U}g$ are path formulas.

Linear temporal logic (LTL), on the other hand, will consist of formulas that have the form $\mathbf{A}f$ 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 $\neg f$, $f \vee g$, $\mathbf{X}f$, and $f\mathbf{U}g$ are path formulas.

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

- W is a set of states or worlds.
- $R \subseteq W \times W$ is the transition relation, which must be total. We write $w_1 \rightarrow w_2$ to indicate that $(w_1, w_2) \in R$.
- $L : W \rightarrow \mathcal{P}(AP)$ 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 $i \geq 0$, $w_i \rightarrow w_{i+1}$. π^i will denote the *suffix* of π 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 π 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. $w \models A \quad \Leftrightarrow \quad A \in L(w)$.
2. $w \models \neg f_1 \quad \Leftrightarrow \quad w \not\models f_1$.
3. $w \models f_1 \vee f_2 \quad \Leftrightarrow \quad w \models f_1$ or $w \models f_2$.
4. $w \models E(g_1) \quad \Leftrightarrow \quad$ there exists a path π starting with w such that $\pi \models g_1$.
5. $\pi \models f_1 \quad \Leftrightarrow \quad w$ is the first state of π and $w \models f_1$.
6. $\pi \models \neg g_1 \quad \Leftrightarrow \quad \pi \not\models g_1$.
7. $\pi \models g_1 \vee g_2 \quad \Leftrightarrow \quad \pi \models g_1$ or $\pi \models g_2$.
8. $\pi \models \mathbf{X}g_1 \quad \Leftrightarrow \quad \pi^1 \models g_1$.
9. $\pi \models g_1 \mathbf{U} g_2 \quad \Leftrightarrow \quad$ there exists a $k \geq 0$ such that $\pi^k \models g_2$ and for all $0 \leq j < k$, $\pi^j \models g_1$.

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

$$f \wedge g \equiv \neg(\neg f \vee \neg g) \quad \mathbf{F}f \equiv \mathit{true}\,\mathbf{U}f \quad \mathbf{A}(f) \equiv \neg\,\mathbf{E}(\neg f) \quad \mathbf{G}f \equiv \neg\mathbf{F}\neg f.$$

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 $\mathbf{A}(\mathbf{FG}p)$. Likewise, there is no LTL formula that is equivalent to the CTL formula $\mathbf{AG}(\mathbf{EF}p)$. The disjunction of these two formulas $\mathbf{A}(\mathbf{FG}p) \vee \mathbf{AG}(\mathbf{EF}p)$ is a CTL* formula that is not expressible in either CTL or LTL.

Although $\mathbf{A}(\mathbf{FG}p)$ cannot be expressed as a CTL formula, $\mathbf{A}(\mathbf{GF}p)$ can be expressed as a CTL formula, and in fact is equivalent to $\mathbf{AG}(\mathbf{AF}p)$. We need to use

this fact in Section 5. To see that it is true assume that $\mathbf{A}(\mathbf{GF}p)$ is false in some state w of a Kripke structure. Hence, there must be a path π such that $\neg p$ holds almost always on π. Let w_1 be the first state on π such that $\neg p$ holds at w_1 and at every state following w_1 on π. Clearly, $\mathbf{AF}p$ does not hold at w_1. Since w_1 is reachable from w, it must be the case that $\mathbf{AG}(\mathbf{AF}p)$ is false at w. Conversely, assume that $\mathbf{AG}(\mathbf{AF}p)$ is false at state w. Thus, $w \models \mathbf{EF}(\mathbf{EG} \neg p)$. It follows that there is a state w_1 reachable from w by a finite path π_0 such that $w_1 \models \mathbf{EG} \neg p$. There must also be a path π_1 starting at w_1 such that $\neg p$ holds globally along π_1. The path π obtained by concatenating π_0 and π_1 starts at w and shows that $\mathbf{A}(\mathbf{GF}p)$ 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 is PSPACE-complete for LTL and for CTL*. In [8], an efficient graph-traversal algorithm is given to solve the model checking problem for CTL.

THEOREM 3.1
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 [5, 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 $W, F \subseteq \mathcal{P}(W)$. A path π is *F-fair iff* the following condition holds: *for each* $G \in F$, *there are infinitely many states on* π *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 3.2
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 [5, 8, 9].

4 Automata on infinite tapes and the logic ECTL

A *Müller 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 $Freq \subseteq \mathcal{P}(St)$ is a set of *frequent* states. An infinite string $\sigma = a_0 a_1, \ldots \in Alph^\omega$ is accepted by M provided the set of states that M enters infinitely often when started in s_0 on string σ 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 $inf(\sigma) = \{s \in St | s = u_k \text{ for infinitely many } k \geq 0\}$, then σ is accepted by M iff $inf(\sigma) \in Freq$. Note that, since Tr is a function, the defined automaton is deterministic. That is, for a given state and a given input at most one transition is defined. This is important for the efficiency of the algorithm described in the next section.

In this paper $Alph = \mathcal{P}(\Sigma)$ will be the set of all possible truth assignments for some nonempty set of proposition symbols Σ. Each truth assignment will be represented by the subset of Σ that is assigned the value true. The elements of Σ 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 Σ to represent subsets of $Alph$. The formula f will represent all 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 $(a \wedge b) \vee (a \wedge \neg b)$ 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 $f_1 \wedge f_2$.

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

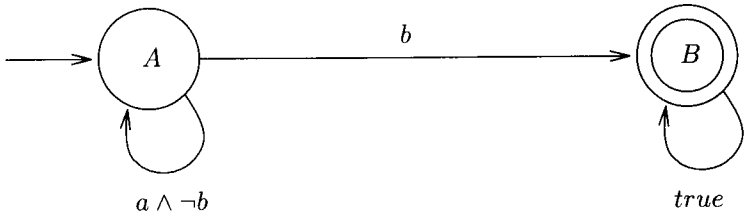

FIG. 1. Müller automaton for until operator.

Figure 2 gives a Müller automaton M_2 over $\Sigma = \{a\}$ for the path formula $\mathbf{FG}a$ *(almost always a)*. Recall from Section 2 that the corresponding CTL* state formula $\mathbf{A}(\mathbf{FG}a)$ 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 $Freq = \{\{B\}\}$ in this case as well.

We are now ready to give the syntax and semantics of ECTL formulas. Let $\{M_i\}$ be a family of Müller automata such that each M_i has $\mathcal{P}(\Sigma_i)$ as its input alphabet where $\Sigma_i = \{a_1^i, \ldots, a_{k_i}^i\}$. 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.

- Every atomic proposition is an ECTL formula.
- If f and g are ECTL formulas, then $\neg f$ and $f \vee g$ are ECTL formulas.
- If $f_1, \ldots, f_{k_i}$ are ECTL formulas, then $\mathbf{E}[M_i](f_1, \ldots, f_{k_i})$ and $\mathbf{A}[M_i](f_1, \ldots, f_{k_i})$ are ECTL formulas.

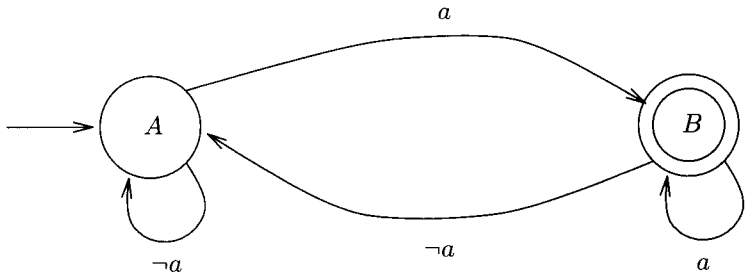

FIG. 2. Müller automaton for *almost always a*.

Thus, if M_1 and M_2 are the Müller automata defined in Figures 1 and 2, a typical ECTL formula might be $\mathbf{A}[M_1](p \vee r, q) \vee \neg \mathbf{E}[M_2](q)$. This formula will hold in a state s if either $(p \vee r)\mathbf{U}q$ holds for every path starting at s, or q is false infinitely often on each path starting at s.

In order to give the semantics of ECTL it is convenient to use the following definition. Let $\pi = w_0, w_1, \ldots$ be a path in a Kripke structure K, and let $C = q_0, q_1, \ldots$ be a sequence over $\mathcal{P}(\{f_1, \ldots, f_k\})$ where $f_1, \ldots, f_k$ are formulas. π *agrees with* C on $\{f_1, \ldots, f_k\}$ if and only if for every $i \geq 0$ and $j = 1, \ldots, k$, $K, w_i \models f_j$ iff $f_j \in q_i$.

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. $M_i(f_1, \ldots, f_{k_i})$ will denote the Müller automaton in which each parameter a_j^i of M_i is replaced by the corresponding formula f_j. With this convention:

- $K, w \models \mathbf{E}[M_i](f_1, \ldots, f_{k_i})$ iff for some path $\pi = w_0, w_1, \ldots$ starting at w in K there is a sequence $C = q_0, q_1, \ldots$ accepted by $M_i(f_1, \ldots, f_{k_i})$ such that π agrees with C on $\{f_1, \ldots, f_{k_i}\}$.
- $K, w \models \mathbf{A}[M_i](f_1, \ldots, f_{k_i})$ iff for every path $\pi = w_0, w_1, \ldots$ starting at w in K there is a sequence $C = q_0, q_1, \ldots$ accepted by $M_i(f_1, \ldots, f_{k_i})$ such that π agrees with C on $\{f_1, \ldots, f_{k_i}\}$.

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

THEOREM 4.1
For every CTL* formula φ there is an ECTL formula ψ which will be true in exactly the same states of a Kripke structure as φ.

The proof of this theorem is postponed to Appendix A.

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 $Freq = \{S_1, \ldots, S_n\} \subseteq \mathcal{P}(V)$. The *E-acceptance problem* for G is to find all of those vertices v such that for some path π starting at v, $inf(\pi)$ is an element of $Freq$. The *A-acceptance problem* for G is to find all of those vertices v such that for every path π starting at v, $inf(\pi)$ is an element of $Freq$.

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

The A-acceptance problem is somewhat more complicated. First we check $\neg\mathbf{EG}true$ with n fairness constraints: **infinitely often** $\neg S_1, \ldots,$ **infinitely often** $\neg S_n$. This procedure will find those vertices v, such that every path starting with v is almost always within some S_l. This test is not sufficient, since a path π might almost always be within S_l but not visit some vertex v_k of S_l infinitely often. In addition, we must insure that for every path π there is a set $S_l \in Freq$ such that $inf(\pi) = S_l$. For each S_l we would like to check the formula

$$\mathbf{A}(\mathbf{FG}S_l \rightarrow [\mathbf{GF}v_1 \wedge \mathbf{GF}v_2 \wedge \ldots \wedge \mathbf{GF}v_{r_l}])$$

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

A vertex v that passes the test will have the property that every path π which is almost always within S_l, but is not almost always in any subset T_i of S_l, visits every vertex in S_l 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_l. To see why this works, consider the case of a particular S_l. Note that if a path π satisfies the fairness constraint **infinitely often** $\neg T_j$, then $inf(\pi)$ is not a subset of T_j. Thus, if path π is fair, $inf(\pi)$ is not contained in any subset $T_1, \ldots, T_{h_l}$ of S_l. Hence, if π is almost always contained within S_l, then it should visit each of the elements of S_l infinitely often. If π 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

$$\mathbf{A}(\mathbf{FG}S_l \rightarrow \mathbf{GF}v_1) \wedge \mathbf{A}(\mathbf{FG}S_l \rightarrow \mathbf{GF}v_2) \wedge \ldots \wedge \mathbf{A}(\mathbf{FG}S_l \rightarrow \mathbf{GF}v_{r_l}).$$

Thus, it is sufficient to be able to check formulas of the form $\mathbf{A}(\mathbf{FG}p \rightarrow \mathbf{GF}q)$ with a series of fairness constraints: **infinitely often** $u_1, \ldots$, **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:

$$\begin{aligned}
\mathbf{A}(\mathbf{FG}p \rightarrow \mathbf{GF}q) &\equiv \mathbf{A}(\neg\mathbf{FG}p \vee \mathbf{GF}q) \\
&\equiv \mathbf{A}((\mathbf{GF}\neg p) \vee \mathbf{GF}q) \\
&\equiv \mathbf{A}(\mathbf{GF}(\neg p \vee q)) \\
&\equiv \mathbf{AG}(\mathbf{AF}(\neg p \vee q))
\end{aligned}$$

The last equivalence follows from the identity $\mathbf{A}(\mathbf{GF}f) \equiv \mathbf{AG}(\mathbf{AF}f)$ 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 $O(|G| \cdot |Freq|^2)$ where $|G|$ and $|Freq|$ are as above.

We would like to use the technique described above to check temporal operators defined in terms of Müller 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 Müller automaton. Let $K = (W, R, L)$ be the Kripke structure and $M = (St, Alph, Tr, s_0, Freq)$ be a *complete* Müller automata, i.e. the transition function of M, Tr, is defined for every element in $St \times Alph$. Assume that $Alph = \mathcal{P}(\Sigma)$. 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

- $w \rightarrow w'$ is a transition of K.
- There is $g \in Alph = \mathcal{P}(\Sigma)$ such that:
  - $Tr(s, g) = s'$ is a transition of M.
  - For every $\alpha \in \Sigma$, $K, w \models \alpha$ iff $\alpha \in g$.

A path p in the product structure $K \times M$ can be decomposed into a path π in K and a path c in M. The path p simulates the behaviour of M on π. More precisely, if c starts at the initial state of M and $inf(c)$ is an element of $Freq$, then path π in K is accepted by the Müller 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 Müller 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 $\mathbf{E}[M](f_1, \ldots, f_m)$ and $\mathbf{A}[M](f_1, \ldots, f_m)$ assuming that $f_1, \ldots, f_m$ are atomic formulas labelling K.

THEOREM 5.1
$K, w \models \mathbf{E}[M](f_1, \ldots, f_m)$ iff $K \times M(f_1, \ldots, f_m)$, $(w, s_0) \models \mathbf{EF}(\mathbf{EG}S_l)$ with fairness constraints: **infinitely often** $s_1, \ldots$, **infinitely often** s_{r_l}, for some $S_l = \{s_1, \ldots, s_{r_l}\} \in Freq$.

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 5.2
$K, w \models \mathbf{A}[M](f_1, \ldots, f_m)$ iff

- $K \times M(f_1, \ldots, f_m), (w, s_0) \models \neg \mathbf{EG}\, true$ with fairness constraints:

$$\textbf{infinitely often} \neg S_1, \ldots, \textbf{infinitely often} \neg S_n.$$

- For every $S_l \in Freq$:

$$K \times M(f_1, \ldots, f_m), (w, s_0) \models \mathbf{A}(\mathbf{FG}S_l \rightarrow [\mathbf{GF}s_1 \wedge \mathbf{GF}s_2 \wedge \ldots \mathbf{GF}s_{r_l}])$$

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

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

The algorithm outlined above will label each state (w, s_0) of $K \times M$ with an **A** 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 labelling algorithm to the subformulas of h, starting with the shortest, most deeply nested and work outward to include all of h. Working in this manner, it is guaranteed that whenever $\mathbf{E}[M](f_1, \ldots, f_m)$ $(\mathbf{A}[M](f_1, \ldots, f_m))$ is considered, $f_1, \ldots, f_m$ have already been processed, i.e., a state $w \in K$ is labelled with f_i, $1 \leq i \leq m$ if and only if $K, w \models f_i$. Defining $\Sigma = \{f_1, \ldots, f_m\}$, the above algorithm can be applied to determine for every $w \in K$ if $\mathbf{E}[M](f_1, \ldots, f_m)$ $(\mathbf{A}[M](f_1, \ldots, f_m))$ is true in w. Thus, the algorithm described above works correctly for every ECTL formula h. Since processing a single Müller automaton M_i with Tr_i as its transition relation and $Freq_i$ as its acceptance set takes time $O(|K| \cdot |Tr_i| \cdot |Freq_i|^2)$ and since h has $length(h)$ different subformulas, the entire algorithm requires $O(|K| \cdot length(h) \cdot |Tr_{max}| \cdot |Freq_{max}|^2)$, where max is the index of the Müller automaton used as an operator in h for which the product $|Tr_i| \cdot |Freq_i|^2$ 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_{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 algorithm 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 non-deterministic versions of the automata discussed in this paper. It is not difficult to show that the model checking problem is PSPACE-hard for ECTL formulas with operators specified by non-deterministic automata. Although an algorithm for this may still be polynomial in the size of the Kripke structure, it will very likely involve some version of the closure construction in [23, 26] 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 Müller 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 the technique described in the paper applies to several different types of automata, it is natural to ask which is best? The translation from a deterministic Büchi automaton to a deterministic Müller automata is exponential in general. The reverse translation (from deterministic Müller 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 Müller automata. L-automata and $\forall$-automata are a step in the right direction since they combine Büchi and Müller 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.

Acknowledgements

We wish to acknowledge the help of T. G. Tang. Also, Michael Kaminski is thanked for useful discussions. The first author was partially supported by NSF Grant MCS-82-16706. The second author was partially supported by a Weizmann postdoctoral fellowship.

References

[1] S. Aggarwal, R. P. Kurshan and K. K. Sabnani, A calculus for protocol specification and validation. In *Protocol Specification, Testing and Verification*, North-Holland, pp. 19–34, 1983.

[2] B. Alpern and F. Schneider, Verifying temporal properties without using temporal logic. Technical Report 85-723, Cornell University Computer Science Department, December 1985.

[3] A. Arnold and P. Crubille, A linear algorithm to solve fixed-point equations on graphs. Technical Report I-8632, Université de Bordeaux I, France, November 1986.

[4] K. S. Brace, R. L. Rudell and R. E. Bryant, Efficient implementation of a BDD package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference.* IEEE Computer Society Press, June 1990.

[5] M. Browne, E. Clarke, D. Dill and B. Mishra, Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, **C-35**(12), 1986.

[6] M. C. Browne, An improved algorithm for the automatic verification of finite state systems using temporal logic. In *Proceedings of the 1986 Conference on Logic in Computer Science*, pp.

260–267, Cambridge, Massachusetts, June 1986.

[7] E. M. Clarke and E. A. Emerson, Synthesis of synchronization skeletons for branching time temporal logic. In *Proceedings of the Workshop on Logic of Programs*, Lecture Notes in Computer Science **131**, Yorktown Heights, NY, Springer-Verlag, 1981.

[8] E. M. Clarke, E. A. Emerson, and A. P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, **8**(2), pp. 244–263, 1986.

[9] D. L. Dill and E. M. Clarke, Automatic verification of asynchronous circuits using temporal logic. *IEE Proceedings*, **133**, pt. E(5), p. 276–82, 1986.

[10] E. A. Emerson and E. M. Clarke, Characterizing properties of parallel programs as fixpoints. In *Proceedings of the Seventh International Colloquium on Automata, Languages and Programming*. Springer Lecture Notes in Computer Science **85**, pp. 169–81, 1981.

[11] E. A. Emerson and J. Y. Halpern, 'Sometimes' and 'Not Never' revisited: On branching time versus linear time. In *Proceedings 10th ACM Symp. on Principles of Programming Languages*, pp. 169–80, 1983.

[12] E. A. Emerson and C. L. Lei, Modalities for model checking: Branching time strikes back. *12th Symposium on Principles of Programming Languages*, New Orleans, La., pp. 89–96, January 1985.

[13] Z. Habasinski, Program verification and Müller automata. Technical University, Poznan.

[14] R. P. Kurshan, Testing containment of ω-regular languages. Technical Report 1121-861010-33-TM, Bell Laboratories Technical Memorandum, 1986.

[15] O. Lichtenstein and A. Pnueli, Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the 12th Annual ACM Symposium on Principles of Programming Languages*, New Orleans, La., pp. 97–107, January 1985.

[16] Z. Manna and A. Pnueli, Specification and verification of concurrent programs by $\forall$-automata. In *Thirteenth ACM Symposium on Principles of Programming Languages*, Munich, pp. 1–12, January 1987.

[17] K. L. McMillan and J. Schwalbe, Formal verification of the Gigamax cache consistency protocol. In *Proceedings of the International Symposium on Shared Memory Multiprocessing*, Japan, pp. 111–134, April 1991.

[18] R. McNaughton, Testing and generating infinite sequences by a finite automaton. *Information and Control*, **9**, 521–530, 1966.

[19] M. Nivat, Behaviors of synchronized systems of processes. Technical Report 81-64, Université Paris 7, November 1981.

[20] J. P. Quielle and J. Sifakis, Specification and verification of concurrent systems in CESAR. In *Proceedings of the Fifth International Symposium in Programming*, pp. 337–51, 1981.

[21] A. P. Sistla and E. M. Clarke, Complexity of propositional temporal logics. *Journal of the Association for Computing Machinery*, **32**(3), 733–749, 1986.

[22] W. Thomas, On chain logic, path logic, and first-order logic over infinite trees. In *Proceedings of the 1987 Conference on Logic in Computer Science*, pp. 245–257, June 1987.

[23] M. Vardi and P. Wolper, An automata-theoretic approach to automatic program verification. In *Proceedings of the Conference on Logic in Computer Science*, Boston, MA, pp 332–44, June 1986.

[24] M. Y. Vardi and P. Wolper, Yet another process logic. In *Proceedings of the Workshop on Logic of Programs*, Lecture Notes in Computer Science, **164**, pp. 501–513. Springer-Verlag, 1983.

[25] P. Wolper, Temporal logic can be more expressive. In *Proceedings of the 22nd Symposium on Foundations of Computer Science*, pp. 340–348, 1981.

[26] P. Wolper, M. Y. Vardi and A. P. Sistla, Reasoning about infinite computation paths. In *Proceedings of the 24th Symposium on Foundations of Computer Science*, pp. 185–194, 1983.

Appendix: Proof of Theorem 4.1

We first present some definitions and lemmas that are needed for the proof. Let *Alph* be a set of state formulas of CTL*. We say that C is a sequence *over Alph* if $C \in Alph^{\omega}$. In the sequel, we

sometimes interpret a path formula with respect to a sequence C rather than with respect to a path π in a Kripke structure. We say that $C = q_0, q_1, \ldots$ satisfies a state formula f (denoted $C \models f$) iff $f \in q_0$. The rest of the definition is identical to the definition of $\models$ over π. We use C^i to denote the suffix of C starting at q_i.

Let f be a path formula of CTL* and let g be a state subformula of f. An occurrence of g in f is *maximal* if this occurrence is not a proper subformula of a proper state-subformula of f. The formula g *is maximal in* f if it has a maximal occurrence in f. $Max(f)$, the set of maximal state-subformulas of f, is inductively defined by:

- If f is a state formula then $Max(f) = \{f\}$.
- If f is a path formula and either $f = \neg f_1$ or $f = \mathbf{X}f_1$, then $Max(f) = Max(f_1)$.
- If f is a path formula and either $f = f_1 \vee f_2$ or $f = f_1 \mathbf{U} f_2$ then $Max(f) = Max(f_1) \cup Max(f_2)$.

When $Max(f) = \{f_1, \ldots, f_k\}$ we sometimes refer to f as $f[f_1, \ldots, f_k]$. Given a set of state formulas $\{\psi_1, \ldots, \psi_k\}$ we denote by $f[\psi_1, \ldots, \psi_k]$ the formula obtained from f by replacing all the maximal occurrences of f_i in f by ψ_i, for $i = 1, \ldots, k$.

LEMMA A.1
Let $f[f_1, \ldots, f_k]$ be a path formula of CTL* with $Max(f) = \{f_1, \ldots, f_k\}$, and let $a_1, \ldots, a_k$ be new proposition names. Then $f[a_1, \ldots, a_k]$ is an LTL formula.

PROOF. Immediate. ∎

LEMMA A.2
For every LTL formula f over the set of atomic propositions AP, there exists a deterministic Müller automaton M_f over $Alph = \mathcal{P}(AP)$ which accepts exactly those sequences over $Alph^\omega$ that satisfy f.

PROOF. In [23] Vardi and Wolper show that for every LTL formula f there is a Büchi automaton B_f that accepts exactly those sequences that satisfy f. By [18] there is a deterministic Müller automaton, M_f, defined over $Alph^\omega$, that accepts exactly the same language as B_f. M_f is the required automaton. ∎

LEMMA A.3
Let $f[f_1, \ldots, f_k]$ be a CTL* path formula with $Max(f) = \{f_1, \ldots, f_k\}$ and let $\{\psi_1, \ldots, \psi_k\}$ and $\{\varphi_1, \ldots, \varphi_k\}$ be two sets of state formulas. Also let $C = q_0, q_1, \ldots$ and $C' = q'_0, q'_1, \ldots$ be two infinite sequences over $\mathcal{P}(\{\psi_1, \ldots, \psi_k\})$ and $\mathcal{P}(\{\varphi_1, \ldots, \varphi_k\})$ respectively, such that for every $i \geq 0$ and $j = 1, \ldots, k$, $\psi_j \in q_i$ iff $\varphi_j \in q'_i$. Then

(a) $C \models f[\psi_1, \ldots, \psi_k]$ iff $C' \models f[\varphi_1, \ldots, \varphi_k]$.

(b) Let π be a path in a Kripke structure K, such that π agrees with C on $\{\psi_1, \ldots, \psi_k\}$, then $C \models f[\psi_1, \ldots, \psi_k]$ iff $K, \pi \models f[\psi_1, \ldots, \psi_k]$.

PROOF. Since the proofs of (a) and (b) are similar, we prove them simultaneously. We use induction on the structure of f to show that for every i,

$$C^i \models f[\psi_1, \ldots, \psi_k] \text{ iff } C'^i \models f[\varphi_1, \ldots, \varphi_k] \text{ iff } K, \pi^i \models f[\psi_1, \ldots, \psi_k].$$

Base case: f is a state formula. Then $Max(f) = \{f\}$, $f[\psi_1] = \psi_1$ and $f[\varphi_1] = \varphi_1$. Thus, (a) and (b) immediately hold.

Inductive step: We show that (a) and (b) hold for $f = g\mathbf{U}h$. The other cases are handled similarly. Given that $f = g\mathbf{U}h$, $Max(f) = Max(g) \cup Max(h)$. Assume that $Max(g) = \{f_{j_1}, \ldots, f_{j_m}\}$ and that $Max(h) = \{f_{l_1}, \ldots, f_{l_n}\}$ where,

$$\{f_1, \ldots, f_k\} = \{f_{j_1}, \ldots, f_{j_m}\} \cup \{f_{l_1}, \ldots, f_{l_n}\}.$$

If $C^i \models (g\mathbf{U}h)[\psi_1, \ldots, \psi_k]$ then there exists a $t \geq i$, such that $C^t \models h[\psi_{l_1}, \ldots, \psi_{l_n}]$ and for all $i \leq r < t$, $C^r \models g[\psi_{j_1}, \ldots, \psi_{j_m}]$. By the inductive hypothesis, $C'^t \models h[\varphi_{l_1}, \ldots, \varphi_{l_n}]$ and $C'^r \models g[\varphi_{j_1}, \ldots, \varphi_{j_m}]$. Moreover, $K, \pi^t \models h[\psi_{l_1}, \ldots, \psi_{l_n}]$ and $K, \pi^r \models g[\psi_{j_1}, \ldots, \psi_{j_m}]$. This implies that, $C'^i \models (g\mathbf{U}h)[\varphi_1, \ldots, \varphi_k]$ and $K, \pi^i \models (g\mathbf{U}h)[\psi_1, \ldots, \psi_k]$. The other implications necessary to complete the proof of the inductive hypothesis proceed along similar lines. ∎

LEMMA A.4
Let $M_f(a_1,\ldots,a_k)$ be an automaton that accepts exactly those sequences over $\mathcal{P}(\{a_1,\ldots,a_k\})$ that satisfy $f[a_1,\ldots,a_k]$. Then, $M_f(\psi_1,\ldots,\psi_k)$ accepts C over $\mathcal{P}(\{\psi_1,\ldots,\psi_k\})$ iff $C \models f[\psi_1,\ldots,\psi_k]$.

PROOF. Let C be a sequence over $\mathcal{P}(\{\psi_1,\ldots,\psi_k\})$ and let C' be a sequence over $\mathcal{P}(\{a_1,\ldots,a_k\})$ obtained from C by replacing everywhere ψ_j by a_j, for $j = 1,\ldots,k$.

Assume C is accepted by $M_f(\psi_1,\ldots,\psi_k)$ by an accepting run $u_0,u_1,\ldots$. Then, $u_0,u_1,\ldots$ is a run in $M_f(a_1,\ldots,a_k)$, that accepts C'. Thus, $C' \models f[a_1,\ldots,a_k]$ and by Lemma A.3 $C \models f[\psi_1,\ldots,\psi_k]$.

Conversely, assume $C \models f[\psi_1,\ldots,\psi_k]$. By Lemma A.3, $C' \models f[a_1,\ldots,a_k]$. Thus, C' is accepted by $M_f(a_1,\ldots,a_k)$ by some accepting run. The same run accepts C in $M_f(\psi_1,\ldots,\psi_k)$. ∎

We are ready now to prove Theorem 4.1. We will show that for every state formula φ of CTL* there is an equivalent ECTL formula ψ, i.e., for every Kripke structure K and a state w in K,

$$K,w \models \varphi \text{ iff } K,w \models \psi.$$

PROOF. We prove the theorem by induction on the depth of the nesting of **E** operators in φ.

Base case: φ does not contain **E** operators. Thus, φ is either an atomic formula or a Boolean combination of atomic formulas. In either case, $\psi = \varphi$.

Inductive step: We show that the claim holds for a state formula φ in which the depth of nesting of **E** operators is at most $n+1$.

Case (a): $\varphi = \mathbf{E}f$, where f is a path formula. Let $Max(f) = \{f_1,\ldots,f_k\}$. Each f_i has at most n nested **E** operators. Thus, by the inductive hypothesis for each f_i there is an ECTL formula ψ_i, equivalent to f_i.

Let $a_1,\ldots,a_k$ be new proposition names. By Lemmas A.1 and A.2, there is a deterministic Müller automaton M_f, over $Alph = \mathcal{P}(\{a_1,\ldots,a_k\})$ that accepts exactly those sequences over $Alph^\omega$ which satisfy $f[a_1,\ldots,a_k]$. We denote this automaton by $M_f(a_1,\ldots,a_k)$. We choose $\psi = \mathbf{E}[M_f](\psi_1,\ldots,\psi_k)$ and show that $\psi \equiv \varphi$.

Assume $K,w \models \mathbf{E}[M_f](\psi_1,\ldots,\psi_k)$. Then, there is a path $\pi = w_0,w_1,\ldots$ starting at w in K and a sequence $C = q_0,q_1,\ldots$ over $\mathcal{P}(\{\psi_1,\ldots,\psi_k\})$ such that C is accepted by $M_f(\psi_1,\ldots,\psi_k)$. Moreover, for every $i \geq 0$ and for all $1 \leq j \leq k$, $K,w_i \models \psi_j$ iff $\psi_j \in q_i$. Since ψ_j and f_j are equivalent, $K,w_i \models f_j$ as well.

Let $C' = q'_0,q'_1,\ldots$ be the sequence obtained from C by replacing ψ_j by f_j in each q_i, for all $j = 1,\ldots,k$. By Lemma A.3, $C \models f[\psi_1,\ldots,\psi_k]$ iff $C' \models f[f_1,\ldots,f_k]$. Since C is accepted by $M_f(\psi_1,\ldots,\psi_k)$, by Lemma A.4 $C \models f[\psi_1,\ldots,\psi_k]$ and therefore $C' \models f[f_1,\ldots,f_k]$. Since π agrees with C' on $\{f_1,\ldots,f_k\}$, $K,\pi \models f[f_1,\ldots,f_k]$. But $f[f_1,\ldots,f_k]$ is identical to f. Thus, $K,w \models \mathbf{E}f$ as required.

Conversely, let $K,w \models \mathbf{E}f$. Then, there exists $\pi = w_0,w_1,\ldots$ starting at w in K such that $K,\pi \models f$. Let $C' = q'_0,q'_1,\ldots$ be a sequence over $\mathcal{P}(\{f_1,\ldots,f_k\})$ such that for every i, $q'_i = \{f_j | f_j \in MAX(f) \text{ and } K,w_i \models f_j\}$. The path π agrees with C' on $\{f_1,\ldots,f_k\}$, therefore $C' \models f[f_1,\ldots,f_k]$.

Let $C = q_0,q_1,\ldots$ be a sequence over $\mathcal{P}(\{\psi_1,\ldots,\psi_k\})$ such that for every i, $q_i = \{\psi_j | f_j \in q'_i\}$. By Lemma A.3, $C \models f[\psi_1,\ldots,\psi_k]$, thus C is accepted by $M_f(\psi_1,\ldots,\psi_k)$. The equivalence between f_j and ψ_j implies that $K,w_i \models \psi_j$ iff $\psi_j \in q_i$, for every $i \geq 0$. Thus, π agrees with C on $\{\psi_1,\ldots,\psi_k\}$ and by the definition of satisfaction of ECTL formulas, $K,w \models \mathbf{E}[M_f](\psi_1,\ldots,\psi_k)$.

Case (b): φ is a Boolean combination of state formulas in which case ψ is the same Boolean combination of the corresponding ECTL formulas. ∎

Received 22 December 1991