Another Look at LTL Model Checking

E. Clarke, O. Grumberg and K. Hamaguchi
February 23, 1994
CMU-CS-94-114
1 Introduction

Over the past thirteen years there has been considerable research on efficient model checking algorithms for branching-time temporal logics like CTL (See [5] for a survey). Verification tools based on these algorithms have discovered non-trivial design errors in sequential circuits and protocols [10] and are now beginning to be used in industry. There has been relatively little research, however, on efficient model checking algorithms for linear-temporal logic (LTL), and practical verification tools are virtually non-existent. In fact, the question of whether it is possible to develop such tools has been argued for many years. Sistla and Clarke [17] showed in 1982 that the model checking problem for LTL was, in general, PSPACE complete. Later, Pnueli and Lichtenstein [14] gave an LTL model checking algorithm that was exponential in the size of the formula, but linear in the size of the model. Based on this result, they argued that the high complexity of LTL model checking might still be acceptable for short formulas. Vardi and Wolper [18] obtained a different algorithm based on \( \omega \)-automata with roughly the same complexity. Unfortunately, the LTL algorithms appeared significantly more difficult to implement. Because of this, very few LTL model checkers were actually constructed. To the best of our knowledge, no experiments were made to determine how the CTL and LTL model checking algorithms actually compared in practice.

In this paper we show how LTL model checking can be reduced to CTL model checking with fairness constraints. We also describe how to construct a symbolic LTL model checker that appears to be quite efficient in practice. In particular, we show how the SMV model checking system developed by McMillan as part of his Ph.D. thesis [16] can be extended to permit LTL specifications. We have developed a translator \( T \) that takes an LTL formula \( f \) and constructs an SMV program \( T(f) \) to build the tableau for \( f \). The tableau construction that we use is similar to the one described in [4]. To check that \( f \) holds for some SMV program \( M \), we combine the text of \( T = T(\neg f) \) with the text of \( M \) to obtain a new SMV program \( P = P(T, M) \). We add CTL fairness constraints to \( P \) in order to make sure that eventualities of the form \( a \mathbf{U} b \) are actually fulfilled (i.e. to eliminate those paths along which \( a \mathbf{U} b \) and \( a \) hold continuously, but \( b \) never holds). By checking an appropriate CTL formula on \( P \) we can find the set \( V_f \) of all of those states \( s \) such that \( f \) holds along every path that begins at \( s \). The projection of \( V_f \) to the state variables of \( M \) gives the set of states where the formula \( f \) holds.

Note that our approach makes it unnecessary to modify SMV (or even understand how SMV is actually implemented). We have evaluated the approach on several standard SMV programs (including Martin's distributed mutual exclusion circuit [15] and the synchronous arbiter described in McMillan's thesis [16]). In order to make sure that the experiments were unbiased, we deliberately chose specifications which could be expressed in both CTL and LTL. The results that we obtained were quite surprising. For the examples we considered, the LTL model checker required at most twice as much time and space as the CTL model checker. Although additional examples still need to be tried, it appears that efficient LTL model checking is possible when the specifications are not excessively complicated. In the full paper we will describe how the same basic approach can be used to extend SMV for testing inclusion between various types of \( \omega \)-automata.
2 Binary Decision Diagrams

Ordered binary decision diagrams (OBDDs) are a canonical form representation for boolean formulas [3]. They are often substantially more compact than traditional normal forms such as conjunctive normal form or disjunctive normal form, and they can be manipulated very efficiently. An OBDD is similar to a binary decision tree, but has the following properties.

- Its structure is a directed acyclic graph rather than a tree.
- A total order is placed on the occurrence of variables as the graph is traversed from root to leaf.
- No two subgraphs in the graph represents the same function.

Bryant showed that given a variable ordering, the OBDD representation for a boolean formula is unique.

We can implement various important logical operations using OBDDs. The function that restricts some argument \( x_i \) of the boolean function \( f \) to a constant value \( b \), denoted by \( f |_{x_i=b} \), can be performed in time which is linear in the size of the original binary decision diagram [3]. The restriction algorithm allows us to compute the OBDD for the formula \( \exists x \exists f \, f |_{x=0} + f |_{x=1} \). All 16 two-argument logical operations can also be implemented efficiently on boolean functions that are represented as OBDDs. The complexity of these operations is linear in the size of the argument OBDDs [3]. Furthermore equivalence checking of two boolean functions can be done in constant time, by using a hash table properly[2].

OBDDs are extremely useful for obtaining concise representations of relations over finite domains [4, 16]. If \( R \) is \( n \)-ary relation over \( \{0, 1\} \), then \( R \) can be represented by the OBDD for its characteristic function

\[
f_R(x_1, \ldots, x_n) = 1 \text{ iff } R(x_1, \ldots, x_n).
\]

Otherwise, let \( R \) be an \( n \)-ary relation over the finite domain \( D \). Using an appropriate binary encoding of \( D \), we can represent \( R \) by an OBDD.

3 Computation Tree Logics

We begin by describing the temporal logic CTL* [8, 9, 12], which can express both linear-time and branching-time properties. In this logic, a path quantifier, either \( \text{A} \) ("for all computation paths") or \( \text{E} \) ("for some computation paths") can prefix an assertion composed of arbitrary combinations of the usual linear-time operators \( \text{G} \) ("always"), \( \text{F} \) ("sometimes"), \( \text{X} \) ("nexttime"), and \( \text{U} \) ("until"). Both Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) are included in CTL*.

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. The syntax of state formulas is given by the following rules:

- If \( p \in AP \), then \( p \) is a state formula.
If \( f \) and \( g \) are state formulas, then \( \neg f \) and \( f \lor g \) are state formulas.

If \( f \) is a path formula, then \( \mathbf{E}(f) \) is a state formula.

Two additional rules are needed to specify the syntax of path formulas:

- If \( f \) is a state formula, then \( f \) is also a path formula.
- If \( f \) and \( g \) are path formulas, then \( \neg f \), \( f \lor g \), \( \mathbf{X} f \), and \( f \mathbf{U} g \) are path formulas.

\( \mathbf{CTL}^\ast \) is the set of state formulas generated by the above rules.

We define the semantics of \( \mathbf{CTL}^\ast \) with respect to a Kripke structure \( M = (S, R, L) \), where \( S \) is the set of states; \( R \subseteq S \times S \) is the transition relation, which must be total (i.e., for all states \( s \in S \) there exists a state \( s' \in S \) such that \( (s, s') \in R \)); and \( L : S \to \mathcal{P}(AP) \) is a function that labels each state with a set of atomic propositions true in that state. In this paper, we assume that all Kripke structures are finite.

A path in \( M \) is an infinite sequence of states, \( \pi = s_0, s_1, \ldots \) such that for every \( i \geq 0 \), \( (s_i, s_{i+1}) \in R \). We use \( \pi^i \) to denote the suffix of \( \pi \) starting at \( s_i \). If \( f \) is a state formula, the notation \( M, s \models f \) means that \( f \) holds at state \( s \) in the Kripke structure \( M \). Similarly, if \( f \) is a path formula, \( M, \pi \models f \) means that \( f \) holds along path \( \pi \) in Kripke structure \( M \).

When the Kripke structure \( M \) is clear from context, we will usually omit it. 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. \( s \models p \iff p \in L(s) \).
2. \( s \models \neg f_1 \iff \models \neg f_1 \).
3. \( s \models f_1 \lor f_2 \iff s \models f_1 \) or \( s \models f_2 \).
4. \( s \models \mathbf{E}(g_1) \iff \) there exists a path \( \pi \) starting with \( s \) such that \( \pi \models g_1 \).
5. \( \pi \models f_1 \iff s \) is the first state of \( \pi \) and \( s \models f_1 \).
6. \( \pi \models \neg g_1 \iff \pi \not\models g_1 \).
7. \( \pi \models g_1 \lor g_2 \iff \pi \models g_1 \) or \( \pi \models g_2 \).
8. \( \pi \models \mathbf{X} g_1 \iff \pi^1 \models g_1 \).
9. \( \pi \models g_1 \mathbf{U} g_2 \iff \) there exists a \( k \geq 0 \) such that \( \pi^k \models g_2 \) and for all \( 0 \leq j < k \), \( \pi^j \models g_1 \).

The following abbreviations are used in writing \( \mathbf{CTL}^\ast \) formulas:

- \( f \land g \equiv \neg (\neg f \lor \neg g) \)
- \( \mathbf{F} f \equiv \text{true} \mathbf{U} f \)
- \( \mathbf{A}(f) \equiv \neg \mathbf{E}(\neg f) \)
- \( \mathbf{G} f \equiv \neg \mathbf{F} \neg f \)

\( \mathbf{CTL} [1, 8] \) is a restricted subset of \( \mathbf{CTL}^\ast \) that permits only branching-time operators—each of the linear-time operators \( \mathbf{G}, \mathbf{F}, \mathbf{X}, \text{ and } \mathbf{U} \) must be immediately preceded by a path quantifier. More precisely, \( \mathbf{CTL} \) is the subset of \( \mathbf{CTL}^\ast \) that is obtained if the following two rules are used to specify the syntax of path formulas:

- If \( f \) and \( g \) are state formulas, then \( \mathbf{X} f \) and \( f \mathbf{U} g \) are path formulas.
- If \( f \) is a path formula, then so is \( \neg f \).
Linear temporal logic (LTL), on the other hand, will consist of formulas that have the form \( A \ f \) where \( f \) is a path formula in which the only state subformulas permitted are atomic propositions. More precisely, a path formula is either:

- an atomic proposition \( p \in AP \).
- If \( f \) and \( g \) are path formulas, then \( \neg f \), \( f \lor g \), \( X \ f \), and \( f \lor g \) are path formulas.

There are eight basic CTL operators: \( AX \), \( EX \), \( AG \), \( EG \), \( AF \), \( EF \), \( AU \) and \( EU \). Each of the eight operators can be expressed in terms of three operators \( EX \), \( EG \), and \( EU \).

4 CTL Model Checking

CTL Model checking is the problem of finding the set of states in a state transition graph where a given CTL formula is true. One approach for solving this problem is a symbolic model checking using an OBDD to represent the transition relation of the graph. Assume that the transition relation is given as a boolean formula \( R(\bar{v}, \bar{v}') \) in terms of current state variables \( \bar{v} = (v_1, \ldots, v_n) \) and next state variables \( \bar{v}' = (v'_1, \ldots, v'_n) \). The algorithm takes a CTL formula \( f \), and the OBDD that represents \( R(\bar{v}, \bar{v}') \). For each subformula \( g \), the algorithm computes the states that satisfy \( g \) in a bottom-up manner. This step is performed by OBDD operations. The algorithm returns an OBDD that represents exactly those states of the system that satisfy the formula \( f \).

Fairness constraints were introduced for checking the correctness of CTL formulas along fair computation paths. A fairness constraint can be an arbitrary set of states, usually described by a 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 then restricted to fair paths. The CTL model checking under given fairness constraints can also be performed using OBDD operations. As will be shown in the next section, LTL model checking can be reduced to CTL model checking under fairness constraints.

5 LTL Model Checking

In this section we consider the model checking problem for linear temporal logic. Let \( A \ f \) be a linear temporal logic formula. Thus, \( f \) is a restricted path formula in which the only state subformulas are atomic propositions. We wish to determine all of those states \( s \in S \) such that \( M, s \models A \ f \). By definition \( M, s \models A \ f \) iff \( M, s \models \neg E \neg f \). Consequently, it is sufficient to be able to check the truth of formulas of the form \( E \ f \) where \( f \) is a restricted path formula. If the Kripke structure is represented explicitly as a state transition graph, this problem is known to be PSPACE-complete [17] in general.

Lichtenstein and Pnueli [14] developed an algorithm for the problem that was linear in the size of the model \( M \) and exponential in the length of the formula \( f \). Although their algorithm was linear in the size of the model, it was still impractical for large examples because of the state explosion problem. As in the case of CTL model checking, representing the transition
relation as an OBDD enables the procedure to be applied to much larger examples. The exponential complexity of their algorithm in terms of formula length is caused by a tableau construction which may require exponential space in the size of the formula.

Burch et. al developed a model checking algorithm for constructing the tableau implicitly [4]. The implicit tableau construction leads to an additional reduction in space and time. We begin with an informal description of the model checking algorithm. Given a formula E f and a Kripke structure M, we construct a special Kripke structure T called the tableau for the path formula f. This structure includes every path that satisfies f. By composing T with M, we find the set of paths that appear in both T and M. A state in M will satisfy E f if and only if it is the start of a path in the composition that satisfies f. The CTL model checking procedure described in Section 4 is used to find these states.

We now describe the construction of the tableau T in detail. Let APf be the set of atomic propositions in f. The tableau associated with f is a structure T = (ST, RT, LT) with APf as its set of atomic propositions. Each state in the tableau is a set of elementary formulas obtained from f. The set of elementary subformulas of f is denoted by el(f) and is defined recursively as follows:

- el(p) = {p} if p E AP.
- el(¬g) = el(g).
- el(g ∨ h) = el(g) U el(h).
- el(X g) = {X g} U el(g).
- el(g U h) = {X(g U h)} U el(g) U el(h).

Thus, the set of states ST of the tableau is P(el(f)). The labeling function LT is defined so that each state is labeled by the set of atomic propositions contained in the state.

In order to construct the transition relation RT, we need an additional function sat that associates with each subformula g of f a set of states in ST. Intuitively, sat(g) will be the set of states that satisfy g.

| sat(g) = {σ | g E σ} where g E el(f). |
| sat(¬g) = {σ | σ ∉ sat(g)}. |
| sat(g ∨ h) = sat(g) U sat(h). |
| sat(g U h) = sat(h) U (sat(g) n sat(X(g U h)))). |

We want the transition relation to have the property that each elementary formula in a state is true in that state. Clearly, if X g is in some state σ, then all the successors of σ should satisfy g. Furthermore, since we are dealing with LTL formulas, if X g is not in σ, then σ should satisfy ¬X g. Hence, no successor of σ should satisfy g. The obvious definition for RT is

\[ R_T(σ, σ') = \bigwedge_{X g} σ \in sat(X g) \iff σ' \in sat(g). \]
Figure 1 gives the tableau for the formula $g = a \cup b$. To reduce the number of edges, we connect two states $\sigma$ and $\sigma'$ with a bidirectional arrow if there is an edge from $\sigma$ to $\sigma'$ and also from $\sigma'$ to $\sigma$. Each subset of $\ell(g)$ is a state of $T$. $sat(Xg) = \{1, 2, 3, 5\}$ since each of these states contains the formula $Xg$. $sat(g) = \{1, 2, 3, 4, 6\}$ since each of these states either contains $b$ or contains $a$ and $Xg$. There is a transition from each state in $sat(Xg)$ to each state in $sat(g)$ and from each state in the complement of $sat(Xg)$ to each state in the complement of $sat(g)$.

Unfortunately, the definition of $RT$ does not guarantee that eventuality properties are fulfilled. We can see this behavior in Figure 1. Although state 3 belongs to $sat(g)$, the path that loops forever in state 3 does not satisfy the formula $g$ since $b$ never holds on that path. Consequently, an additional condition is necessary in order to identify those paths along which $f$ holds. A path $\pi$ that starts from a state $\sigma \in sat(f)$ will satisfy $f$ if and only if

- For every subformula $g \cup h$ of $f$ and for every state $\sigma$ on $\pi$, if $\sigma \in sat(g \cup h)$ then either $\sigma \in sat(h)$ or there is a later state $\tau$ on $\pi$ such that $\tau \in sat(h)$.

In order to state the key property of the tableau construction, we must introduce some new notation. Let $\pi = s_0, s_1, \ldots$ be a path in a Kripke structure $M$, then $label(\pi) = L(s_0), L(s_1), \ldots$. Let $l = l_0, l_1, \ldots$ be a sequence of subsets of some set $\Sigma$ and let $\Sigma' \subseteq \Sigma$. The restriction of $l$ to $\Sigma'$, denoted by $l|_{\Sigma'}$, is the sequence $l'_0, l'_1, \ldots$ where $l'_i = l_i \cap \Sigma'$ for every $i \geq 0$. The following theorem makes precise the intuitive claim that $T$ includes every path which satisfies $f$.

**Theorem 1** Let $T$ be the tableau for the path formula $f$. Then, for every Kripke structure $M$ and every path $\pi'$ of $M$, if $M, \pi' \models f$ then there is a path $\pi$ in $T$ that starts in a state in $sat(f)$, such that $label(\pi')|_{AP_f} = label(\pi)$.

We prove this theorem in the Appendix.
Figure 2: Kripke Structure M

Next, we want to compute the product $P = (S, R, L)$ of the tableau $T = (S_T, R_T, L_T)$ and the Kripke structure $M = (S_M, R_M, L_M)$.

- $S = \{(\sigma, \sigma') \mid \sigma \in S_T, \sigma' \in S_M \text{ and } L_M(\sigma') \cap AP_T = L_T(\sigma)\}$.
- $R((\sigma, \sigma'), (\tau, \tau'))$ iff $R_T(\sigma, \tau)$ and $R_M(\sigma', \tau')$.
- $L((\sigma, \sigma')) = L_T(\sigma)$.

$P$ contains exactly the sequences $\pi''$ for which there are paths $\pi$ in $T$ and $\pi'$ in $M$ such that $\text{label}(\pi'') = \text{label}(\pi) = \text{label}(\pi')|_{AP_T}$. We extend the function $\text{sat}$ to be defined over the set of states of the product $P$ by $(\sigma, \sigma') \in \text{sat}(g)$ if and only if $\sigma \in \text{sat}(g)$.

We next apply CTL model checking and find the set of all states $V$ in $P$, $V \subseteq \text{sat}(f)$, that satisfy $\text{EG} \text{true}$ with the fairness constraints

$$\{\text{sat}(- (g \text{ U } h) \vee h) \mid g \text{ U } h \text{ occurs in } f\}. \tag{1}$$

Each of the states in $V$ is in $\text{sat}(f)$. Moreover, it is the start of an infinite path that satisfies all of the fairness constraints. These paths have the property that no subformula $g \text{ U } h$ holds almost always on the path while $h$ remains false. The correctness of our construction is summarized by the following theorem.

**Theorem 2** $M, \sigma' \models \text{E} \text{f}$ if and only if there is a state $\sigma$ in $T$ such that $(\sigma, \sigma') \in \text{sat}(f)$ and $P, (\sigma, \sigma') \models \text{EG} \text{true}$ under fairness constraints $\{\text{sat}(- (g \text{ U } h) \vee h) \mid g \text{ U } h \text{ occurs in } f\}$.

The proof of this theorem is also given in the Appendix.

To illustrate this construction, we check the formula $g = a \text{ U } b$ on the Kripke structure $M$ in Figure 2. The tableau $T$ for this formula is given in Figure 1. If we compute the product $P$ as described above, we obtain the Kripke structure shown in Figure 3. We use the CTL model checking algorithm to find the set $V$ of states in $\text{sat}(g)$ that satisfy the formula $\text{EG} \text{true}$ with the fairness constraint $\text{sat}(- (a \text{ U } b) \vee b)$. It is easy to see that the fairness constraint corresponds to the following set of states $\{(2', 4'), (5, 3'), (7, 1'), (6, 2'), (1, 2')\}$. Thus, every state in Figure 3 satisfies $\text{EG} \text{true}$. However, only $(2', 4')$, $(3, 1')$, $(1, 2')$ are $(6, 2')$ are in $\text{sat}(g)$, so the states $1'$, $2'$, and $4'$ of $M$ satisfy $Eg = E[a \text{ U } b]$.

We now describe how the above procedure can be implemented using OBDDs. We assume that the transition relation for $M$ is represented by an OBDD as in the previous
In order to represent the transition relation for \( T \) in terms of OBDDs, we associate with each elementary formula \( g \) a state variable \( v_g \). We describe the transition relation \( R_T \) as a boolean formula in terms of two copies \( \bar{v} \) and \( v' \) of the state variables. The boolean formula is converted to an OBDD to obtain a concise representation of the tableau. When the composition \( P \) is constructed, it is convenient to separate out the state variables that appear in \( AP_f \). The symbol \( \bar{p} \) will be used to denote a boolean vector that assigns truth values to these state variables. Thus, each state in \( S_T \) will be represented by a pair \((\bar{p}, \bar{r})\), where \( \bar{r} \) is a boolean vector that assigns values to the state variables that appear in the tableau but not in \( AP_f \). A state in \( S_M \) will be denoted by a pair \((\bar{p}, \bar{q})\) where \( \bar{q} \) is a boolean vector that assigns values to the state variables of \( M \) which are not mentioned in \( f \). Thus, the transition relation \( R_P \) for the product of the two Kripke structures will be given by

\[
R_P(\bar{p}, \bar{q}, \bar{r}, \bar{p}', \bar{q}', \bar{r}') = R_T(\bar{p}, \bar{r}, \bar{p}', \bar{r}') \land R_M(\bar{p}, \bar{q}, \bar{p}', \bar{q}')
\]

We use the symbolic model checking algorithm that handles fairness constraints to find the set of states \( V \) that satisfy \( EG \) true with the fairness constraints given in (1). The states in \( V \) are represented by boolean vectors of the form \((\bar{p}, \bar{q}, \bar{r})\). Thus, a state \((\bar{p}, \bar{q})\) in \( M \) satisfies \( Ef \) if and only if there exists \( \bar{r} \) such that \((\bar{p}, \bar{q}, \bar{r}) \in V \) and \((\bar{p}, \bar{r}) \in sat(f)\).

### 6 LTL Model Checking Using the SMV Model Checker

As stated in Section 5, LTL model checking can be reduced to CTL model checking under fairness constraints. If the tableau and the fairness constraints for a given LTL formula are represented implicitly as boolean formulas, we can perform symbolic LTL model checking using an existing symbolic model checker for CTL. We have developed a translator that enables the SMV model checker to handle LTL formulas. For a given LTL formula, the translator generates an SMV program for the corresponding tableau and fairness constraints. We can perform symbolic LTL model checking using the resulting SMV program. In this section, we describe how the translator works.

We begin with a brief description of the SMV model checker. SMV is a tool for checking that finite-state systems satisfy specifications given in CTL. It uses the OBDD-based symbolic model checking algorithm in Section 4. The language component of SMV is used to describe complex finite-state systems. Figure 4 shows an SMV program for the Kripke structure in Figure 2 and an specification \( A(a U b) \). This example illustrates the basic features
of SMV that are needed to explain the translation procedure. The syntax and semantics of the complete language are given in McMillan’s thesis [16].

SMV users can decompose the description of a complex finite-state system into modules. Module definitions begin with the keyword MODULE. The module main is the top-level module. (The example in Figure 4 contains a single module; however, our translator can handle programs with multiple modules.) Variables are declared using the keyword VAR. In the example, a and b are boolean variables (line 3-4). The TRANS statements are used to define transitions of the model (lines 5-8). In the TRANS statements, next (g) is obtained from g by replacing each state variable v in g by the corresponding next state variable v'. For example, next(a & !b) means a' & ~b' where a' are b' are the next state variables for a and b, respectively. Thus, each TRANS statement determines a propositional formula that relates the original state variables and the next state variables. The transition relation for an SMV program is obtained by taking the conjunction of these formulas. CTL formulas are declared as specifications using the keyword SPEC (line 9).

Next, we describe the translation algorithm. Suppose that we have an SMV program with an LTL formula A f, instead of a CTL formula, as its specification. As stated in Section 5, it is sufficient to handle a formula E ~f. The translator replaces A f with an SMV description of the tableau and the fairness constraints for ~f. The translation of the SMV program in Figure 5 is shown in Figure 6. The translation follows the general procedure outlined in Section 5:

1. Associate a state variable with each elementary formula of ~f.
2. Represent the transition relation of the tableau for \( \neg f \) as a boolean formula in terms of the state variables.

3. Represent fairness constraints as boolean formulas in terms of the state variables.

4. Generate a CTL specification.

In the first step, the formula \( f \) is negated and expanded to a formula in which the only operators are \( \lor, \neg, \Box, \mathcal{U} \). The parse tree of \( \neg f \) is traversed to find its elementary formulas. If a node associated with formula \( Xg \) (or \( g \mathcal{U} h \)) is visited, then the corresponding elementary formula \( Xg \) (or \( X(g \mathcal{U} h) \)) is stored in the list \( el.list \). The translator declares a new variable \( ELXg \) for each formula \( Xg \) in the list \( el.list \). Since atomic propositions are already declared in the original SMV program, they are not declared again.

In order to generate descriptions for the transition relation and the fairness constraints, we have to construct the characteristic function \( S_h \) of \( sat(h) \) for each subformula or elementary formula \( h \) in \( \neg f \). The translator builds these functions using a DEFINE statement\({}^1\). The translator traverses the parse tree of \( \neg f \), and generates the appropriate SMV statements at each node.

\[
S_h := \begin{cases} 
  p & \text{if } p \text{ is an atomic proposition.} \\
  EL_h & \text{if } h \text{ is elementary formula } Xg \text{ in } el.list. \\
  \neg g & \text{if } h = \neg g. \\
  S_{g_1} \lor S_{g_2} & \text{if } h = g_1 \lor g_2. \\
  S_{g_2} \land (S_{g_1} \land X^g(g \mathcal{U} h)) & \text{if } h = g_1 \mathcal{U} g_2.
\end{cases}
\]

The transition relation can be described in terms of the characteristic functions as follows:

\[
\bigwedge_{Xg \in el(f)} SXg(\bar{v}) \Leftrightarrow S_g(\bar{v}')
\]

The expression \( S_g(\bar{v}') \) is represented in SMV by \texttt{next}(\( S_g \)). The translator constructs a formula \( SXg = \texttt{next}(S_g) \) for each \( Xg \) in \( el.list \). These formulas are combined in a TRANS statement to give the transition relation for the tableau.

\texttt{TRANS}

( \( SX_{g_1} = \texttt{next}(S_{g_1}) \) ) &
( \( SX_{g_2} = \texttt{next}(S_{g_2}) \) ) &
\vdots
( \( SX_{g_N} = \texttt{next}(S_{g_N}) \) )

Likewise, the translator traverses the parse tree and generates an SMV FAIRNESS constraint for each node associated with a formula of form \( g \mathcal{U} h \):

\[\text{This statement associates a symbol with an SMV expression. When the symbol appears in the program, it is replaced with the expression.}\]
Finally, the translator generates an SMV SPEC statement. From Theorem 2, it is clear that the formula $E \neg f$ can be checked using the the specification $S_{-f} \land EG True$. Thus, in order to check the LTL formula $A f = \neg E \neg f$, the translator constructs an SMV SPEC statement for $\neg (S_{-f} \land EG True)$.

We illustrate the translation procedure by applying it to the simple example in Figure 4. The result of this procedure is shown in Figure 7. The statements in lines 1 through 8 come from the original SMV program, while the statements in lines 9 through 19 are generated by the tableau construction for $a \cup b$. The translation procedure first determines that $a$, $b$ and $X(a \cup b)$ are elementary formulas and causes the state variable $EL_{X_a \cup b}$ to be declared on line 10. Next, the DEFINE statement in lines 12 through 16 is constructed for the characteristic functions of sat($a$), sat($b$), sat($X(a \cup b)$), sat($a \cup b$) and sat($\neg a \cup b$). The Trans statement in line 17 causes the transition relation for the tableau to be constructed, and line 18 contains the fairness constraint for $a \cup b$. Finally, the specification to be checked is given by the 'SPEC' statement in line 19.

7 Experimental Results

This section describes the experimental results that we obtained for symbolic LTL model checking. In order to compare the performance of LTL model checking with CTL model checking, we used two sequential circuit designs whose specifications can be described in both LTL and CTL.

The first example is a distributed mutual exclusion (DME) circuit designed by Alain Martin[15]. The DME circuit is a speed-independent token ring, which consists of identical arbiter cells. A user of the DME circuit obtains exclusive access to the resource via request and acknowledge signals. We assume arbitrary delay for all gates in the circuit. Each gate is modeled as a finite-state machine that non-deterministically decides either to recompute its output or remain unchanged. We verify the correctness of the following two specifications:

1. (Safety) No two users are acknowledged simultaneously.
2. (Liveness) All requests are eventually acknowledged.

The safety specification is given by the formula

$$AG \bigwedge_{1 \leq i < j \leq n} \neg(ack_i \land ack_j),$$

where $ack_i$ means that user $i$ is acknowledged. This formula is both an LTL formula and a CTL formula. In the experiments for this specification, infinite delays are allowed at each gate. In other words, the output value of each gate can remain unchanged forever.

Next, we verify that requests are eventually acknowledged. We only check this specification with respect to a single user (user 1). In this case the LTL specification has the form:

$$AG(req_1 \rightarrow Fack_1)$$
```
--- Kripke structure

MODULE ...
MODULE ...
MODULE main ...

--- Tableau for f

VAR -- new variables

ELX_{g_1} : boolean;
ELX_{g_2} : boolean;
...
ELX_{g_N} : boolean;

DEFINE -- characteristic function
S_{h_1} := ...;
S_{h_2} := ...;
...
S_{h_M} := ...;

TRANS -- transition relation
( SX_{g_1} = next (S_{g_1}) ) &
( SX_{g_2} = next (S_{g_2}) ) &
...
( SX_{g_N} = next (S_{g_N}) )

--- fairness constraints
FAIRNESS !S_{g_1} U H' | S_{h_1}'
FAIRNESS !S_{g_2} U H' | S_{h_2}'
...
FAIRNESS !S_{g_M} U H' | S_{h_M}'

--- new specification
SPEC !(S_\neg f & EG true)
```

Figure 6: Translator output for SMV program
1 MODULE main -- simple program

2 VAR

3 a: boolean;
4 b: boolean;

5 TRANS (a & !b) -> next(!(a & !b))
6 TRANS (a & b) -> next(a & !b)
7 TRANS (!a & b) -> next(!a & b)
8 TRANS (!a & !b) -> next(!a & b)

9 VAR

10 EL_X_a_U_b : boolean;

11 DEFINE

12 S_a := a;
13 S_b := b;
14 S_X_a_U_b := EL_X_a_U_b;
15 S_a_U_b := S_b \lor (S_a & S_X_a_U_b);
16 S_NOT_a_U_b := !S_a_U_b;

17 TRANS S_X_a_U_b = next(S_a_U_b)

18 FAIRNESS !S_a_U_b \lor b

19 SPEC !(S_NOT_a_U_b & EG true)

Figure 7: Translator output for simple SMV program

This formula is equivalent to the CTL formula:

$$\text{AG}(\text{req} \rightarrow \text{AF} \text{ack}_1)$$

If infinite delays are allowed at each gate, these formulas are not true. In order to overcome this problem we use a fairness constraint which ensures that the output of the gate is reevaluated infinitely often.

SMV provides several options to perform model checking. We verified the circuit using the following approach.

- A single OBDD is constructed for the transition relation of the circuit.

- The reachable states of the circuit are determined, and evaluation of the CTL operators is restricted to these states.
Table 1 summarizes the experimental results for the safety specification, and Table 2 summarizes the results for the liveness specification. The columns show the number of the cells (#cell), the maximum number of OBDD nodes used at any given time (#nodes), the run time on SPARC station 10 (time), the size of the transition relation in OBDD nodes (trans.) and the number of the reachable states (#reachable states). In the experiment for the safety specification, we observe that the number of reachable states for LTL model checking is twice as large as for CTL model checking. The increase in allocated OBDD nodes and run time is less than 10%. In the experiment for the liveness specification, the number of the reachable states is four times larger for LTL model checking, while the increase in space and time is 1.5–3 times larger.

The second example is a synchronous bus arbiter which is described in McMillan’s thesis [16]. This circuit is composed of a daisy chain of identical arbiter cells. The requester with the highest priority receives an acknowledgement from the arbiter under normal operation, while a round-robin scheme is applied when the bus traffic becomes very heavy. Each cell is modeled by a deterministic machine, so the whole arbiter circuit is also a deterministic
Table 3: Safety specification for the synchronous arbiter

<table>
<thead>
<tr>
<th>#cell</th>
<th>#nodes</th>
<th>#time/sec</th>
<th>trans.</th>
<th>#reachable states</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CTL</td>
<td>LTL</td>
<td></td>
<td>CTL</td>
</tr>
<tr>
<td>3</td>
<td>384</td>
<td>734</td>
<td>0.08</td>
<td>0.1</td>
</tr>
<tr>
<td>4</td>
<td>654</td>
<td>1279</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>5</td>
<td>987</td>
<td>1913</td>
<td>0.11</td>
<td>0.15</td>
</tr>
<tr>
<td>6</td>
<td>1383</td>
<td>2628</td>
<td>0.13</td>
<td>0.18</td>
</tr>
<tr>
<td>7</td>
<td>1842</td>
<td>3424</td>
<td>0.16</td>
<td>0.21</td>
</tr>
<tr>
<td>8</td>
<td>2364</td>
<td>4301</td>
<td>0.16</td>
<td>0.26</td>
</tr>
<tr>
<td>9</td>
<td>2949</td>
<td>5259</td>
<td>0.16</td>
<td>0.33</td>
</tr>
<tr>
<td>10</td>
<td>3597</td>
<td>6298</td>
<td>0.21</td>
<td>0.33</td>
</tr>
<tr>
<td>11</td>
<td>4308</td>
<td>7418</td>
<td>0.21</td>
<td>0.41</td>
</tr>
<tr>
<td>12</td>
<td>5082</td>
<td>8619</td>
<td>0.31</td>
<td>0.45</td>
</tr>
</tbody>
</table>

1. (Safety) No two users are acknowledged simultaneously.

2. (Liveness) All requests are eventually acknowledged.

In fact, exactly the same LTL and CTL specifications can be used.

In the experiments using SMV, we used the options to construct single transition relations, and to compute reachable states before model checking. Table 3 shows the experimental results for the safety specification and Table 4 shows the results for the liveness specification. For the safety specification we observe that the number of reachable states for LTL model checking is twice as large as for CTL model checking. The number of the allocated OBDD nodes and run time both increase by a factor of 1.5. In the second experiment, the number of the reachable states is four times larger for LTL model checking. The amount of space and time that is required is 1.5–2 times larger.

8 Directions for Future Research

Certainly the most important thing that remains to be done is to try additional examples. Based on the two examples that we have considered in detail so far, it appears that efficient LTL model checking is possible when the formula that is being checked is not excessively complicated. This does not mean that LTL will take the place of CTL in model checking applications. Many other problems, like testing inclusion and equivalence between various types omega-automata [7], can also be reduced to CTL model checking. LTL, on the other hand, does not appear to have this flexibility. Moreover, in many of the applications of model checking to verification, it is important to be able to assert the existence of a path that satisfies some property. For example, absence of deadlock might be expressed by the
Table 4: Liveness specification for the synchronous arbiter

<table>
<thead>
<tr>
<th>#cell</th>
<th>#nodes</th>
<th>#time(sec)</th>
<th>trans.</th>
<th>#reachable states</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CTL</td>
<td>LTL</td>
<td>CTL</td>
<td>LTL</td>
</tr>
<tr>
<td>3</td>
<td>996</td>
<td>2159</td>
<td>0.10</td>
<td>0.26</td>
</tr>
<tr>
<td>4</td>
<td>1531</td>
<td>3137</td>
<td>0.20</td>
<td>0.36</td>
</tr>
<tr>
<td>5</td>
<td>2155</td>
<td>4254</td>
<td>0.38</td>
<td>0.43</td>
</tr>
<tr>
<td>6</td>
<td>2867</td>
<td>5483</td>
<td>0.43</td>
<td>0.48</td>
</tr>
<tr>
<td>7</td>
<td>3667</td>
<td>6820</td>
<td>0.48</td>
<td>0.61</td>
</tr>
<tr>
<td>8</td>
<td>4555</td>
<td>8266</td>
<td>0.53</td>
<td>0.81</td>
</tr>
<tr>
<td>9</td>
<td>5531</td>
<td>9821</td>
<td>0.71</td>
<td>1.01</td>
</tr>
<tr>
<td>10</td>
<td>6595</td>
<td>10000</td>
<td>0.83</td>
<td>1.23</td>
</tr>
<tr>
<td>11</td>
<td>7747</td>
<td>10001</td>
<td>1.00</td>
<td>1.46</td>
</tr>
<tr>
<td>12</td>
<td>8987</td>
<td>10052</td>
<td>1.16</td>
<td>1.71</td>
</tr>
</tbody>
</table>

CTL formula $\text{AG EF } \text{start}$ (Regardless of what state the program enters, there exists a computation leading back to the $\text{start}$ state). Neither this formula nor its negation can be expressed in LTL [6], so LTL model checking techniques cannot be used to decide whether the formula is true or not. Ideally, it should be possible to reason about linear-time and branching-time properties in the same logic (say, CTL*). We believe this goal can potentially be realized by extending the techniques discussed in this paper. Emerson and Lei [13] have shown how to reduce CTL* model checking to LTL model checking. If the transformation outlined in this paper can be extended to incorporate their reduction, then it should be possible to develop a model checker that can handle both types of properties.

Appendix

We prove Theorem 1 and Theorem 2 of Section 5.

**Theorem 1** Let $T$ be the tableau for the path formula $f$. Then, for every Kripke structure $M$ and every path $\pi'$ of $M$, if $M, \pi' \models f$ then there is a path $\pi$ in $T$ that starts in a state in $\text{sat}(f)$, such that $\text{label}(\pi') \mid_{\text{AP}} = \text{label}(\pi)$.

In order to prove this theorem, we need the following two lemmas. In the remainder of this section, $\pi' = s_0 s_1' \ldots$ represents a path in $M$. We denote the suffix of $\pi'$ starting from the state $s_i$ as $\pi'_i$, i.e., $\pi'_i = s_i s_{i+1}' \ldots$. For the path $\pi'_i$, we define $s_i = \{ \psi | \psi \in \text{el}(f) \}$ and $M, \pi'_i \models \psi$. Note that $s_i$ is a state in $T$.

**Lemma 3** For all $g \in \text{sub}(f) \cup \text{el}(f)$, $M, \pi'_i \models g$ if and only if $s_i \in \text{sat}(g)$.

**Proof.** The proof proceeds by induction on the structure of the formula.

1. Case $g \in \text{el}(f)$. By the definition of $s_i$, it is easy to see that $M, \pi'_i \models g$ if and only if $g \in s_i$. By the definition of $\text{sat}$, $g \in s_i$ if and only if $s_i \in \text{sat}(g)$. 

16
2. Case $g = \neg g_1$ and $g = g_1 \lor g_2$. By the induction hypothesis and the definition of $\text{sat}$, it is easy to prove these cases.

3. Case $g = g_1 \lor g_2$. By the definition of $\text{U}$, $M, \pi_i' \models g_1 \lor g_2$ if and only if $M, \pi_i' \models g_2$ or $(M, \pi_i' \models g_1$ and $M, \pi_i' \models X(g_1 \lor g_2))$. By the induction hypothesis and the definition of $s_i$, $M, \pi_i' \models g_2$ or $(M, \pi_i' \models g_1$ and $M, \pi_i' \models X(g_1 \lor g_2))$ if and only if $s_i \in \text{sat}(g_2) \land (s_i \in \text{sat}(g_1) \land s_i \in \text{sat}(X(g_1 \lor g_2)))$. By the definition of $\text{sat}$, $s_i \in \text{sat}(g_2) \land (s_i \in \text{sat}(g_1) \land s_i \in \text{sat}(X(g_1 \lor g_2)))$ if and only if $s_i \in \text{sat}(g_1 \lor g_2)$.

**Lemma 4** Given $\pi' = s_0s_1' \cdots$ and $s_i$ as above, then $\pi = s_0s_1 \cdots$ is a path in $T$.

**Proof.** Clearly, for all $i$, $s_i \in S_T$. By Lemma 3 and the definition of $X$, it is easy to see the following relation: $s_i \in \text{sat}(Xg)$ if and only if $M, \pi_i' \models Xg$ if and only if $M, \pi_{i+1}' \models g$ if and only if $s_{i+1} \in \text{sat}(g)$. By the definition of $R_T$, if $s_i \in \text{sat}(Xg)$ then $s_i \in \text{sat}(X(g_1 \lor g_2))$ if and only if $s_i \in \text{sat}(g_1 \lor g_2)$. Therefore $\pi = s_0s_1 \cdots$ is a path in $T$. □

**Proof of Theorem 1.** Suppose that, for a path $\pi'$ in $M, \pi' \models f$. By Lemma 4, we can find a path $\pi = s_0s_1 \cdots$ in $T$. By Lemma 3, $s_0 \in \text{sat}(f)$. By the definition of $s_i$, $L(s_i') |_{AP_f} = L_T(s_i)$. and thus $\text{label}(\pi') |_{AP_f} = \text{label}(\pi)$. This leads to Theorem 1. □

**Theorem 2** $M, \sigma' \models E \neg f$ if and only if there is a state $\sigma$ in $T$ such that $(\sigma, \sigma') \in \text{sat}(f)$ and $P, (\sigma, \sigma') \models \text{EG True}$ under the fairness constraints given in (1).

In order to prove this theorem, we need the following three lemmas.

**Lemma 5** Given $\pi = s_0s_1 \cdots$ where $s_i$ is defined as above, then $\pi \models \text{G True}$ under the the fairness constraints given in (1).

**Proof.** In order to show that $\pi \models \text{G True}$ under the fairness constraints, we need to prove that, for every subformula $g \text{ U h}$ of $f$, there are infinitely many states $s_i$ on $\pi$ such that $s_i \in \text{sat}(\neg(g \text{ U h}) \lor h)$. Suppose not, then there exists $i_0$ such that, for all $i \geq i_0$, $s_i \not\in \text{sat}(\neg(g \text{ U h}) \lor h)$. Thus $s_i \in \text{sat}(g \text{ U h})$ and $s_i \not\in \text{sat}(h)$. By Lemma 3, for all $i \geq i_0$, $\pi_i' \models g \text{ U h}$ and $\pi_{i} \not\models h$. Since $\pi_i' \models g \text{ U h}$ means $\pi_j' \models h$ for some $j \geq i$, this leads to a contradiction. □

It is easy to see the next lemma.

**Lemma 6** $\pi'' = (s_0, s_0')(s_1, s_1') \cdots$ is a path in $P$ with $L_P((s_i, s_i')) = L_T(s_i)$ for all $i \geq 0$ if and only if there exist a path $\pi = s_0s_1 \cdots$ in $T$ and a path $\pi' = s_0's_1' \cdots$ in $M$ with $L_T(s_i) = L_M(s_i) |_{AP_f}$ for all $i \geq 0$.

**Lemma 7** Assume that, for all $k \geq j$, $s_k \in \text{sat}(g_1)$ if and only if $\pi_k \models g_1$. $s_k \in \text{sat}(g_2)$ if and only if $\pi_k \models g_2$.

If $\pi_j \not\models g_1 \text{ U g}_2$ and $s_j \in \text{sat}(g_1 \text{ U g}_2)$, then, for all $k \geq j$, $\pi_k \not\models g_1 \text{ U g}_2$ and $s_k \in \text{sat}(g_1 \text{ U g}_2)$.

**Proof.** First we prove that, if $s_j \in \text{sat}(g_1 \text{ U g}_2)$ and $\pi_j \not\models g_1 \text{ U g}_2$, then $s_{j+1} \in \text{sat}(g_1 \text{ U g}_2)$ and $\pi_{j+1} \not\models g_1 \text{ U g}_2$. From the definition of $\text{sat}$, $\text{sat}(g_1 \text{ U g}_2)$ implies $s_j \in \text{sat}(g_2)$ or $(s_j \in \text{sat}(g_1)$ and $s_j \in \text{sat}(X(g_1 \text{ U g}_2)))$. From the assumptions and the definition of $R_T$, it follows that:

$$\pi_j \models g_2 \text{ or } (\pi_j \models g_1 \text{ and } s_{j+1} \in \text{sat}(g_1 \text{ U g}_2)). \quad (2)$$
Since \( \pi_j \not\models g_1 \cup g_2 \) implies \( \pi_j \not\models g_2 \), (2) leads to the following:

\[
\pi_j \models g_1 \text{ and } s_{j+1} \in \text{sat}(g_1 \cup g_2).
\] (3)

Since \( \pi_j \models g_1 \) from (3) and \( \pi_j \not\models g_1 \cup g_2 \) from the assumption, we can also get \( \pi_{j+1} \not\models g_1 \cup g_2 \). Similarly we can get, for all \( k = j + 2, j + 3, j + 4 \ldots \), \( s_k \in \text{sat}(g_1 \cup g_2) \) and \( \pi_k \not\models g_1 \cup g_2 \). □

**Lemma 8** Let \( \pi \models \mathbf{G} True \) under the fairness constraints, then \( T, \pi \models f \) if and only if \( s_0 \in \text{sat}(f) \).

**Proof.** By induction on the structure of the formula, we prove, for each \( g \in \text{sub}(f) \cup \ell(f), \forall j : T, \pi_j \models g \) if and only if \( s_j \in \text{sat}(g) \).

1. Case \( g = p \in AP \). By the definition of \( s \), and the definition of \( \text{sat} \), it is easy to see the following relation: \( \pi_j \models p \) if and only if \( p \in L_T(s_j) \) if and only if \( p \in s_j \) if and only if \( s_j \in \text{sat}(p) \).

2. Case \( g = \neg g_1 \) and \( g = g_1 \lor g_2 \). By the induction hypothesis and the definition of \( \neg \) and \( \lor \), it is easy to prove these cases.

3. Case \( g = X g_1 \). By the definition of \( R_T \) and the induction hypothesis, we can see the following relation: \( s_j \in \text{sat}(X g_1) \) if and only if \( s_{j+1} \in \text{sat}(g) \) if and only if \( \pi_{j+1} \models g \) if and only if \( \pi_j \models X g_1 \).

4. Case \( g = g_1 \cup g_2 \). (\( \Rightarrow \)) Assume that \( \pi_j \models g_1 \cup g_2 \), then for some \( l \geq j \), \( \pi_i \models g_2 \) and for all \( j \leq i < l \), \( \pi_i \models g_1 \). By the induction hypothesis, \( s_i \in \text{sat}(g_2) \) and therefore \( s_i \in \text{sat}(g_1 \cup g_2) \). By the definition of \( R_T \), it follows that \( s_{i-1} \in \text{sat}(X(g_1 \cup g_2)) \). But \( \pi_{i-1} \models g_1 \), so, by induction \( s_{i-1} \in \text{sat}(g_1) \) and therefore \( s_{i-1} \in \text{sat}(g_1 \cup g_2) \). By induction on \((l - j)\) we eventually get \( s_j \in \text{sat}(g_1 \cup g_2) \).

(\( \Leftarrow \)) Suppose that \( s_j \in \text{sat}(g_1 \cup g_2) \) and \( \pi_j \not\models g_1 \cup g_2 \). By Lemma 7, for all \( k \geq j \), \( s_k \in \text{sat}(g_1 \cup g_2) \) and \( \pi_k \not\models g_1 \cup g_2 \). This implies that \( s_k \not\models g_2 \), and thus \( s_k \not\in \text{sat}(g_2) \) from the induction hypothesis. Consequently \( s_k \in \text{sat}(g_1 \cup g_2) \) and \( s_k \not\in \text{sat}(g_2) \) for all \( k \geq j \). This leads to a contradiction, because \( \pi \models \mathbf{G} True \) guarantees that there are infinitely many states \( s_k \) such that \( s_k \in \text{sat}(\neg(g_1 \cup g_2) \lor g_2) \). Therefore if \( s_j \in \text{sat}(g_1 \cup g_2) \), then \( \pi_j \models g_1 \cup g_2 \). □

**Proof of Theorem 2.** (\( \Rightarrow \)) Since \( M, s_0 \models \mathbf{E} f \), then \( \exists \pi' \models f \). By Theorem 1 and Lemma 5, we can prove, for \( \pi \in T \), \( \pi \models \mathbf{G} True \) and \( \text{label}(\pi) = \text{label}(\pi') |_{AP} \). By Lemma 6, there is a path \( \pi'' \) in \( P \) such that \( \text{label}(\pi'') = \text{label}(\pi) \). Since \( \text{label}(\pi) = \text{label}(\pi') |_{AP} \) and \( \pi' \models f \), we can see \( \pi \models f \). Also since \( \pi \models \mathbf{G} True \), by Lemma 8 \( s_0 \in \text{sat}(f) \). Thus \( (s_0, s_0') \in \text{sat}(f) \). Since \( \text{label}(\pi) = \text{label}(\pi'') \) and \( \pi \models \mathbf{G} True \), it is clear that \( \pi'' \models \mathbf{G} True \). Therefore \( P, (s_0, s_0') \models \mathbf{EG} True \).

(\( \Leftarrow \)) Since \( (s_0, s_0') \in \text{sat}(f) \) and \( P, (s_0, s_0') \models \mathbf{EG} True \), then \( \exists \pi'' \models \mathbf{G} True \). By Lemma 6, there exist paths \( \pi \in T \) and \( \pi' \in M \) such that \( \text{label}(\pi'') = \text{label}(\pi) = \text{label}(\pi') |_{AP} \). Since \( \pi'' \models \mathbf{G} True \) and \( \text{label}(\pi) = \text{label}(\pi'') \), we can see \( \pi \models \mathbf{G} True \). Since \( (s_0, s_0') \in \text{sat}(f) \), \( s_0 \in \text{sat}(f) \). From Lemma 8, \( \pi \models f \). Since \( \text{label}(\pi) = \text{label}(\pi') |_{AP} \), \( \pi' \models f \). Therefore \( M, s_0' \models \mathbf{E} f \). □
References


Another Look at LTL Model Checking

E. Clarke, O. Grumberg and K. Hamaguchi

February 23, 1994
CMU-CS-94-114

Abstract

We show how LTL model checking can be reduced to CTL model checking with fairness constraints. Using this reduction, we also describe how to construct a symbolic LTL model checker that appears to be quite efficient in practice. In particular, we show how the SMV model checking system developed by McMillan [16] can be extended to permit LTL specifications. The results that we have obtained are quite surprising. For the examples we considered, the LTL model checker required at most twice as much time and space as the CTL model checker. Although additional examples still need to be tried, it appears that efficient LTL model checking is possible when the specifications are not excessively complicated.
Keywords: automatic verification, temporal logic, model checking, binary decision diagrams