Efficient Decision Procedures for Model Checking of Linear Time Logic Properties*

Roderick Bloem[1], Kavita Ravi[2], and Fabio Somenzi[1]

[1] Department of Electrical and Computer Engineering
University of Colorado, Boulder, CO, 80309-0425
{Roderick.Bloem,Fabio}@Colorado.EDU
[2] Cadence Design Systems
New Providence, NJ, 07974-1143
kravi@cadence.com

Abstract. We propose an algorithm for LTL model checking based on the classification of the automata and on guided symbolic search. Like most current methods for LTL model checking, our algorithm starts with a tableau construction and uses a model checker for CTL with fairness constraints to prove the existence of fair paths. However, we classify the tableaux according to their structure, and use efficient decision procedures for each class. Guided search applies hints to constrain the transition relation during fixpoint computations. Each fixpoint is thus translated into a sequence of fixpoints that are often much easier to compute than the original one. Our preliminary experimental results suggest that the new algorithm for LTL is quite efficient. In fact, for properties that can be expressed in both CTL and LTL, the algorithm is competitive with the CTL model checking algorithm.

1 Introduction

Successful application of model checking requires strategies to bridge the gap between the size of the models and the capacity of the model checkers. Abstraction closes the gap from above by eliminating unnecessary detail from the models and decomposing complex proofs into sequences of simpler ones. Abstraction is fundamental to the practical use of model checking. It is important, however, to close the gap also from below—by increasing the capacity of the model checkers. Indeed, too much reliance on abstraction inevitably means too much reliance on manual intervention, which in turns entails low productivity and exposure to errors.

The symbolic approach to model checking (BDD-based [4,26] and, more recently, SAT-based [2]) addresses the complexity issue by representing models, sets of states, and paths as solutions to equations. Though not uniformly superior to the approach based on the explicit representation of states, symbolic model checking can deal with many more states and transitions. On the other hand, it has proved hard to predict whether a model will exceed the memory and time limits imposed on a given experiment: Whereas models with as many as 5000 state variables have been analyzed successfully without any abstraction, other models with 30 state variables turn out to be intractable. In this paper

* This work was supported in part by SRC contract 98-DJ-620.

N. Halbwachs and D. Peled (Eds.): CAV'99, LNCS 1633, pp. 222–235, 1999.
© Springer-Verlag Berlin Heidelberg 1999

we propose techniques that improve the performance and robustness of BDD-based model checking algorithms for linear time properties [24,34].

Model checking for Linear Time Logic (LTL) is usually based on converting the property into a Büchi automaton (or tableau), composing the automaton and the model, and finally checking for emptiness of the language of the composed system. The last step can be performed by CTL model checking with fairness constraints [6]. In the context of this general strategy, our contribution is twofold: First, we propose a classification of the automata obtained by translation of the properties; our classification refines the one proposed in [20] to three types: general, weak, and terminal automata. We show that applying a specific decision procedure to each class results in an algorithm that is superior to the standard one both in theory and in practice. Different tableau constructions produce automata that may differ according to our classification. We adopt the procedure of [15] because it tends to produce automata that are amenable to more efficient decision procedures.

Converting properties into automata and applying specialized decision procedures based on the structure of the automaton tends to reduce the number of fixpoints that must be computed by the model checker. If the number of fixpoints is reduced to one, on-the-fly model checking can be easily applied [1]. In Section 5 we show that this sometimes produces substantial savings in memory and CPU time, even when comparing to CTL model checking. In general, our experiments confirm and strengthen the observation of [20] about the efficiency of LTL model checking.

Our second contribution is the extension of guided symbolic search from reachability analysis [32] to LTL model checking. Guided symbolic search applies constraints to the transition relation of the model to make the computation of fixpoints more efficient. The constraints are eventually lifted, so that the result of the computation does not differ from the the one of the conventional approach. However, by exploring the state space not in strict breadth-first fashion, guided search can be substantially more efficient than conventional fixpoint computations. The constraints can be seen as *hints* on the order in which transitions should be explored. Effective hints can be derived with only a limited understanding of the behavior of the model subjected to verification. In this paper we show how to apply hints to both least and greatest fixpoint computations. The asymmetry in the two computations is another reason for reducing the number of fixpoints when translating LTL properties.

The rest of this paper is organized as follows. Section 2 reviews the background material. Section 3 discusses the classification of the automata derived from LTL properties and the decision procedures for each class, while Section 4 deals with the application of guided symbolic search to model checking. Section 5 presents our preliminary experimental results, and Section 6 summarizes, outlines future work, and concludes.

2 Preliminaries

2.1 Linear Time Model Checking

We adopt the positive normal form (a.k.a. negation normal form) for the specification of LTL. Given a set of atomic propositions A, the standard boolean connectives, and the

temporal operators X (*next time*), U (*until*) and R (*releases*), LTL formulae in positive normal form are defined as follows:

- **true**, **false**, the atomic propositions, and their negations are formulae;
- if φ and ψ are formulae, then so are $\varphi \vee \psi$, $\varphi \wedge \psi$, $\mathsf{X}\,\varphi$, $\varphi \,\mathsf{U}\, \psi$, and $\varphi \,\mathsf{R}\, \psi$.

It is customary to define two additional operators: $\mathsf{F}\,\varphi$ abbreviates $\mathbf{true}\ \mathsf{U}\ \varphi$ and $\mathsf{G}\,\varphi$ abbreviates **false** $\mathsf{R}\ \varphi$. The boolean connectives $\rightarrow$ and $\leftrightarrow$ are also defined as abbreviations in the usual way.

We define the semantics of LTL with respect to a Kripke structure $\langle S, T, S_0, A, L\rangle$, where S is the set of states, $T \subseteq S \times S$ is the transition relation, $S_0 \subseteq S$ is the set of initial states, A is the set of atomic propositions A, and $L : S \rightarrow 2^A$ is the labeling function. The transition relation is assumed to be *complete*; that is, every state has at least one successor. An infinite path π in M is an infinite sequence $s_0, s_1, \ldots$ such that $(s_i, s_{i+1}) \in T$ for $i \geq 0$. We denote by π^i the suffix of π starting at s_i. The satisfaction of an LTL formula along path π of M is defined as follows.

$\pi \models \mathbf{true}$

$\pi \not\models \mathbf{false}$

$\pi \models \varphi$ iff $\varphi \in L(s_0)$ for $\varphi \in A$

$\pi \models \neg\varphi$ iff $\varphi \notin L(s_0)$ for $\varphi \in A$

$\pi \models \varphi \vee \psi$ iff $\pi \models \varphi$ or $\pi \models \psi$

$\pi \models \varphi \wedge \psi$ iff $\pi \models \varphi$ and $\pi \models \psi$

$\pi \models \mathsf{X}\,\varphi$ iff $\pi^1 \models \varphi$

$\pi \models \varphi \,\mathsf{U}\, \psi$ iff there exists $i \geq 0$ such that $\pi^i \models \psi$, and for all j, $0 \leq j < i$, $\pi^j \models \varphi$

$\pi \models \varphi \,\mathsf{R}\, \psi$ iff for all $i \geq 0$ $\pi^i \models \psi$, or there exists j, $0 \leq j < i$, such that $\pi^j \models \varphi$

A formula is *satisfied* in a Kripke structure M if it is satisfied along a path of M such that $s_0 \in S_0$. A formula is *valid* in a Kripke structure M if it is satisfied along all paths of M such that $s_0 \in S_0$. Given an LTL formula in positive normal form, its negation can be computed by recursively applying De Morgan's Laws and the following identities: $\neg\mathsf{X}\,\varphi = \mathsf{X}\,\neg\varphi$, and $\neg(\varphi \,\mathsf{U}\, \psi) = \neg\varphi \,\mathsf{R}\, \neg\psi$. Writing the negation in positive normal form does not change the length of the formula $|\varphi|$, if one assumes that for an atomic proposition p, $|p| = |\neg p|$.[1] Therefore we can efficiently solve the validity problem for φ by checking the satisfiability of $\neg\varphi$. This is the approach that we adopt in the sequel.

Model checking of linear time property φ is usually accomplished by constructing a Büchi automaton $B_{\neg\varphi}$ from the formula $\neg\varphi$. This automaton is often referred to as the *tableau* of the formula; it accepts runs that visit sets of *fair states* infinitely often. The product of the automaton $B_{\neg\varphi}$ and the model M is then analyzed to see if it contains a so-called *fair cycle*. A fair cycle reachable from the initial states signals satisfaction of $\neg\varphi$; hence, it is a counterexample to the validity of φ in M. When explicit enumeration is used, the run time of the model checking algorithm is linear in the size of the model and exponential in the length of the formula.

In the rest of this paper we shall have occasion to refer to logics other than LTL. Computational Tree Logic (CTL) is a branching time logic. Temporal operators are always preceded by universal or existential path quantifiers in CTL formulae. The expressiveness of CTL is not comparable to that of LTL: Properties like $\mathsf{AG}\,\mathsf{EF}\,p$ have no equivalent in LTL, while $\mathsf{F}\,\mathsf{G}\,p$ (fairness) is not expressible in CTL. Model checkers for

[1] This assumption is valid for BDD-based model checkers.

CTL usually allow the user to specify fairness constraints separately from the property. Both LTL and CTL are subsumed by CTL*, which is in turn subsumed by the $L\mu_2$ fragment of the μ-calculus. The reader interested in the formal definition and a detailed analysis of these logics is referred to [12].

2.2 Symbolic Model Checking

The main difficulty to model checking comes from the size of the state space S. This is typically true also of LTL model checking, in spite of the exponential dependence of the runtime on the length of the formula, because the formulae of interest are usually short. *Symbolic model checking* [6,26,2] addresses this concern by representing sets of states *implicitly* via their *characteristic functions*. In this paper we consider BDD-based symbolic model checking, in which Binary Decision Diagrams [4] are used to represent the characteristic functions. Although almost all boolean functions have exponentially sized BDDs [27], symbolic model checkers have been successful on problems that vastly exceed the capacity of explicit enumeration algorithms. BDDs can be manipulated efficiently; in particular, algorithms have been devised for the computation of all the successors (*image* computation) or predecessors (*pre-image* computation) of a set of states according to a given transition relation [10,5,14,31].

Symbolic model checking algorithms for various logics are based on the computation of fixpoints by repeated image or pre-image computations. In the relational μ-calculus (see, for instance, [26]), the computation of the states reachable from S_0 is expressed by the formulae

$$\begin{aligned} \mathsf{EY}\, p &= \lambda y . \exists x . T(x,y) \wedge p(x) \\ \mathsf{Rch}\, S_0 &= \mu Z . S_0 \vee \mathsf{EY}\, Z \ , \end{aligned}$$

which prescribe a sequence of image computations. Symbolic LTL model checking, on the other hand, is normally based on the algorithm of Emerson and Lei [13] for the $L\mu_2$ fragment of μ-calculus. If the acceptance condition of the automaton is described by a set of fair states, C, the set of states from which a fair cycle can be reached, Fair, is given by:

$$\begin{aligned} \mathsf{EX}\, p &= \lambda x . \exists y . T(x,y) \wedge p(y) \\ \mathsf{E} p \,\mathsf{U}\, q &= \mu Z . q \vee (p \wedge \mathsf{EX}\, Z) \\ \mathsf{Fair} &= \nu Z . \mathsf{EX}\, \mathsf{E}(Z \,\mathsf{U}\, (Z \wedge C)) \ . \end{aligned}$$

If $\mathsf{Fair} \cap S_0 \neq \emptyset$ for $M \times B_{\neg\varphi}$, then there is a fair path, and the LTL formula φ is not valid in M. The observation that Fair is the set of states that satisfy the CTL [9] formula $\mathsf{EG}\,\mathbf{true}$ under the fairness constraint C has led to the use of symbolic model checkers for fair CTL in LTL model checking [6,8].

Many fixpoint computations used in symbolic model checking, including the two just mentioned, can be formulated both in terms of image computations (forward traversal of the state space) and in terms of pre-image computations (backward traversal). In recent times considerable attention has been devoted to the relative efficiency of the alternative formulations [19,18,17,28].

Besides the direction of traversal of the state space, an important factor affecting the efficiency of symbolic model checking algorithms is the presence of multiple fixpoints, especially if nested. Thus, the computation of fair states is intrinsically more difficult than the computation of reachable states.

3 Classification of Tableaux

As outlined in Section 2.1, a linear time property can be checked by converting its negation into a Büchi automaton called the tableau of the property, composing the tableau with the model, and checking language emptiness. The last step of this procedure involves the computation of nested fixpoints and is therefore potentially expensive. The question naturally arises as to whether LTL model checkers can compete with CTL model checkers for those properties that can be expressed in both logics. Kupferman and Vardi [20,21] call these properties *branchable* and observe that many of them translate into tableaux with special structure. They claim that an appropriate variant of the LTL model checking algorithm can then achieve efficiency comparable to that of CTL model checkers. In this section we recall the classification of [20] and refine it in a natural yet effective way.

Different tableau construction procedures have been proposed in the literature [24, 34,6,8,15]. Even though the automata produced by these procedures for a given formula obviously accept the same language, they have different structures, and therefore are not equivalent from the point of view of the classification we propose. We discuss this issue at the end of this section.

A Büchi automaton is a quintuple $\langle \Sigma, Q, Q_0, \delta, F \rangle$, where Σ is the input alphabet, Q is the finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is the acceptance condition. An input word is accepted iff there is a run of the automaton on that word that visits F infinitely often. We assume that the transition function is complete, that is $\delta(q, \sigma) \neq \emptyset$ for all $q \in Q$ and $\sigma \in \Sigma$.

A Büchi automaton is *weak* [20,29] iff there exists a partition of Q into $Q_1, \ldots, Q_n$ such that each Q_i is either contained in F or disjoint from it; in addition, the blocks of the partition are partially ordered so that the transitions of the automaton never move from Q_i to Q_j unless $Q_i \leq Q_j$.

Theorem 1. *The language of a weak Büchi automaton $\mathcal{A}$ is empty iff $\mathcal{A} \models \neg \mathsf{EFEG}\, F$.*

Proof. A run of a weak Büchi automaton that leaves a block Q_i of the partition cannot enter it again. Hence, the only way for a run to visit a fair state infinitely often is to eventually be confined inside one $Q_i \subseteq F$. Such a run therefore is a witness for $\mathsf{EFEG}\, F$. Conversely, if $\mathcal{A}, s_0 \models \mathsf{EFEG}\, F$ for some $s_0 \in S_0$, then there is a fair run in $\mathcal{A}$ and its language is not empty. □

A Büchi automaton is *terminal* iff it is weak and the blocks of the partition contained in F are maximal elements of the partial order.

Theorem 2. *The language of a terminal Büchi automaton $\mathcal{A}$ is empty iff $\mathcal{A} \models \neg \mathsf{EF}\, F$.*

Proof. An accepting run in a terminal Büchi automaton must reach a maximal block Q_i of the partition, otherwise no fair state is visited. Conversely, a run that reaches a maximal block can always be extended to a fair run, because δ is complete. It is therefore necessary and sufficient for a fair run to reach a fair state. □

Theorems 1 and 2 provide the foundation for our model checking strategy. The CTL model checker is used to prove $\neg \mathsf{EG}\, \mathbf{true}$ under the fairness constraint F, $\neg \mathsf{EFEG}\, F$, or $\neg \mathsf{EF}\, \neg F$, depending on the classification of the automaton. Correctness follows from the fact that the composition of the model and a terminal Büchi automaton is a terminal Büchi automaton, and the composition of the model and a weak Büchi automaton is weak. Checking whether an automaton is weak or terminal can be done in polynomial time. The checking of the properties can be carried out by either backward or forward analysis. Forward analysis applied to terminal automata corresponds to reachability analysis.

As pointed out in [16], the Emerson-Lei algorithm, which is quadratic in the size of the state space, is often much slower in practice than reachability analysis and CTL model checking, which are linear. Our classification has the desirable effect of using the more efficient algorithms when possible.

Several variants of the tableau construction have been proposed in the literature. All are based on the identity $\varphi \,\mathsf{U}\, \psi = \psi \vee (\varphi \wedge \mathsf{X}\,(\varphi \,\mathsf{U}\, \psi))$, but they differ in the details. Figure 1 shows the tableaux produced by the procedures of [8] (left) and [15] (center) for the formula $f = p \,\mathsf{U}\, q$. It also shows a variant of the tableau of [15] (right) that has labels on the arcs instead of the states and a complete transition function.

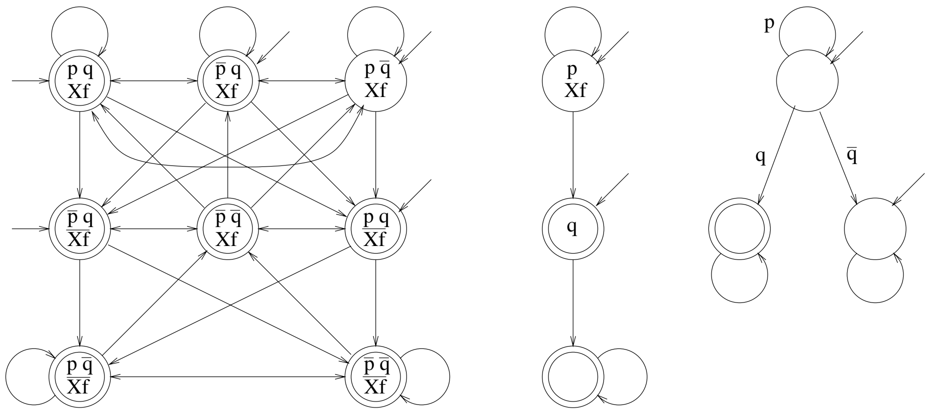

Fig. 1. Tableaux for $p \,\mathsf{U}\, q$. Fair states are indicated by double circles, initial states have an extra incoming arrow, and negation is indicated by an overbar.

The construction of [8] identifies the elementary subformulae of the given formula f (p, q, and $\mathsf{X}\, f$ in our example) and creates one state in the tableau for each combination of elementary subformulae of f. It then adds a transition (s, s') if, for all elementary

subformulae g, either $\mathsf{X}\, g$ holds in s and g holds in s', or $\mathsf{X}\, g$ does not hold in s and g does not hold in s'.

The procedure of [15] creates states of the tableau on the fly, starting with a node that represents f and adding nodes according to the syntactic structure of the the formula.

The disparity in number of states is the most visible difference between the results of the two constructions, but it is of little consequence on the efficiency of symbolic model checking. In fact, since every atomic proposition appearing in f also appears—possibly negated—in the label of each state, the left tableau shares the state variables for p and q with the model and only requires one additional bit for $\mathsf{X}\, f$. The other two automata need two extra bits.

Another difference between the two constructions is of greater import for the efficiency of the model checker. The automaton on the left of Fig. 1 is not weak, unlike the other two (which are indeed terminal). The approach of adding all possible transitions to the automaton at once, instead of adding them as they become needed tends to create more paths in the tableau; hence, it tends to prevent the partial ordering of the states required for weakness.

Therefore, we use the construction of [15] modified to yield automata with labels on the arcs and complete transition functions as in [20]. These modifications allow us to easily express the automata in Verilog that we use as input language for our experiments. It should be noted that our choice is not optimal from the point of view of the number of state variables and transitions of the composition of the model and the property automaton.

4 Guided Search in Model Checking

4.1 Guided Search for the Computation of Least Fixpoints

In [32] it is shown that symbolic reachability analysis, and hence invariant checking, can be substantially sped up by applying *hints*. The hints are predicates on the inputs or state variables of the model. Their effect is to inhibit some transitions; it is obtained by conjoining the hints and the transition relation. Several hints may be applied in sequence. Therefore the computation of the reachable states is decomposed in the computation of a sequence of fixpoints—one for each hint.

Theorem 3. *Given a sequence of monotonic functionals* $\tau_1, \tau_2, \ldots, \tau_k$ *such that* $\tau_i \leq \tau_k$ *for* $0 < i < k$*, the sequence* $\rho_0, \rho_1, \ldots, \rho_k$ *of least fixpoints defined by*

$$\begin{aligned} \rho_0 &= 0 \\ \rho_i &= \mu X . \rho_{i-1} \vee \tau_i(X), \quad 0 < i \leq k \end{aligned}$$

monotonically converges to $\rho = \mu X . \tau_k(X)$*; that is,* $\rho_0 \leq \rho_1 \leq \cdots \leq \rho_k = \rho$.

Proof. We prove by induction that $\rho_i \leq \rho$. The basis is trivially established ($\rho_0 = 0 \leq \rho$). For the inductive step we have:

$$\rho_i \;=\; \mu X . \rho_{i-1} \vee \tau_i(X) \;\leq\; \mu X . \rho_{i-1} \vee \tau_k(X) \;=\; \mu X . \tau_k(X) \enspace ,$$

where the last equality follows from the inductive hypothesis and the properties of fixpoints. The sequence is clearly monotonic, and for $i = k$ the inductive step shows that $\rho_k = \rho$. □

Decomposing the computation of a least fixpoint may have two main advantages; both are based on the fact that an appropriately chosen τ_i (i.e., a properly chosen hint) may make the computation of ρ_i orders of magnitude faster than the direct computation of ρ [32]. The first advantage is that one may not need to compute the whole sequence of fixpoints. For instance, a state that violates an invariant may be contained in ρ_1, in which case the rest of the computation can be avoided. The second advantage applies also to cases in which the computation of ρ must be carried to completion. Indeed, it may be much more efficient to compute ρ from ρ_{k-1} than to compute it directly. In this latter respect symbolic guided search differs from explicit guided search [35], in which guidance is only used to accelerate the detection of states where invariants do not hold.

In [32] evidence is presented in support of the claim that finding good hints requires understanding of the system to be verified at a level comparable to that required to write functional tests for it. In this paper we extend the use of hints from invariant checking to LTL.

4.2 Guided Search for the Computation of Greatest Fixpoints

The method presented in [32] applies to the computation of least fixpoints. Least fixpoints suffice to check invariants, but not for the properties of more expressive logics. In this section we therefore describe the extension of guided symbolic search to the computation of greatest fixpoints. Hints produce underapproximations of the transition relation; therefore, guided symbolic search can complement known methods [22,25,7, 23], when both lower bounds and upper bounds are required [30].

The main objective of the works just cited is to prove the desired property of the system on a simplified model. Our objective is complementary: We want to speed up the computation of the fixpoints for a given model, by addressing the computational bottlenecks; for instance, image computations that are too slow and memory consuming because of poor quantification schedule [14,31].

Another possible reason for using underapproximations in greatest fixpoints is to deal with nested fixpoints. If a greatest fixpoint is nested in a least fixpoint, or vice versa, then by using underapproximations for both computations, one obtains an underapproximation of the result if either computation is restricted to a prefix of the sequence. (For instance, $\rho_0, \dots, \rho_j$ for $j < k$ in the case of a least fixpoint.)

Theorem 4. *Given a sequence of monotonic functionals $\tau_1, \tau_2, \dots, \tau_k$ such that $\tau_i \leq \tau_k$ for $0 < i < k$, the sequence $\eta_0, \eta_1, \dots, \eta_k$ defined by*

$$\begin{aligned} \eta_0 &= 0 \\ \eta_i &= \nu X . \eta_{i-1} \vee \tau_i(X), \quad 0 < i \leq k \end{aligned}$$

monotonically converges to $\eta = \nu X . \tau_k(X)$; that is, $\eta_0 \leq \eta_1 \leq \dots \leq \eta_k = \eta$.

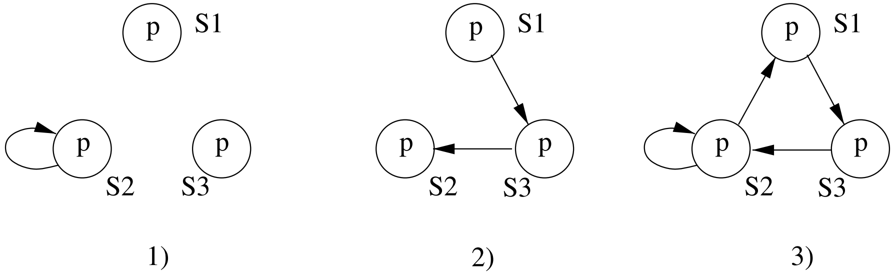

Fig. 2. Greatest fixpoint computation illustrating $\eta_i > \eta_{i-1} \vee \nu X.\tau_i(X)$.

The proof is along the lines of the proof of Theorem 3. It should be observed that $\eta_i \geq \eta_{i-1} \vee \nu X.\tau_i(X)$, and that the inequality can be strict, as shown in the example of Fig. 2. In the example $k = 3$. The rightmost graph can be though of as the original system, and the other two graphs as the systems obtained by applying hints. Let $\mathsf{EX}_i X$ compute the predecessors of the states in X in the graph shown in Part i of Fig. 2. Let $\tau_i = \lambda X.p \wedge \mathsf{EX}_i X$. With these definitions, η is the set of states along infinite paths where p always holds. One can verify that $\eta_2 = \{S1, S2, S3\} = \eta > \{S2\} = \eta_1 \vee \nu X.\tau_2(X)$.

Theorem 4 can speed up model checking in two ways. If the greatest fixpoint being computed is the outermost fixpoint, as in $\mathsf{EG}\,\mathbf{true}$, then we may not have to compute all k fixpoints in the sequence if there is indeed a fair cycle. In these cases, a procedure that yields a large subset of the fixpoint at a small fraction of the computational cost is desirable. This application of Theorem 4 is complementary to the approach of [16], which works best when there are no fair cycles.

In cases where convergence must be proved, Theorem 4 can still speed up computation if R^+ and η^+ are given upper bounds on the reachable states and the greatest fixpoint, and $\eta_i \geq R^+ \wedge \eta^+$ for $i < k$. This case occurs in the example of Fig. 2, in which η_2 equals the trivial upper bound on η given by S itself; hence, η_3 needs not be computed. Finally, if $R^+ \wedge \eta^+ > \eta_i$ for all $i < k$, it may still be that $\eta_{k-1} = \eta$. In this case, the last iteration of the computation of η_k can be skipped because η_{i-1} is a lower bound on η_i.

Comparing Theorems 3 and 4, the following observations are in order. First of all, Theorem 3 can be extended by not insisting on the convergence of all the fixpoints except the last. This is possible because the j-th iterate of the computation of ρ_i contains ρ_{i-1} and is contained in ρ_i. However, this is not the case of the greatest fixpoint computation. Another important difference is that knowledge of ρ_{k-1} helps the computation of ρ_k more than knowledge of η_{k-1} helps the computation of η_k. In practice, application of Theorem 3 tends to be more effective than application of Theorem 4 when convergence must be reached.

5 Experimental Results

In this section we present preliminary experimental results that we have obtained with VIS 1.3 [3], extended to accept hints. CPU times are measured on an IBM Intellistation running Linux with an Intel Pentium II 400 MHz CPU and 1 GB of RAM.

We consider properties that can be expressed in both LTL and CTL. This gives us a means to contrast our decision procedure against the CTL decision procedure. The results can be found in Table 1. The leftmost column gives the model and the type of property. *Soap* is a model of a token-passing algorithm for distributed mutual exclusion [11]. The token is passed along a spanning tree of a network of processors. For this model we checked a liveness property of the form $\mathsf{G}\,(p \rightarrow \mathsf{F}\,q)$ and two safety properties of the form $\mathsf{G}\,(p \rightarrow \mathsf{X}\,(q\;\mathsf{R}\;r))$ expressing the requirement that access to the resource is not granted to a processor unless the processor has requested it. The first of these two properties fails (error in the specification), while the second passes. *Gcd* is a model of a circuit computing the greatest common divisor of two integers; *Pcell* is a model of a production cell; *Palu* is a three-stage pipeline with an ALU and a register file; *Fpmult* is a floating point multiplier; finally, *NullModem* is a circuit to check the correctness of a simple UART and of the handshaking between the UART and a processor.

We ran all the applicable algorithms for each formula and compared CPU time and memory requirements. The table also reports the number of pre-images (EX) and images (EY) computed by each approach. These numbers indicate whether model checking used pure forward analysis (0 pre-images), pure backward analysis (0 images) or a combination of the two: reachability analysis followed by backward model checking. For each formula we checked, one of these three methods was clearly superior to the other two. Results are reported for that method. Some of the results could have been improved by using forward CTL [19], but for uniformity all experiments have been conducted using backward CTL only.

The importance of choosing the right algorithm is underlined by the results of Table 1, which were obtained with the same fixed variable order for all the runs of a given model and without hints. They confirm the observation of [20] about the comparable efficiency of CTL and LTL model checkers when the right algorithms are used for the latter. Notice, however, that experiments performed with dynamic variable reordering enabled may yield quite different results, simply because the orders end up being different. For formulae that fail, the ability to check the property on-the-fly may result in a substantial advantage to our algorithm.

Nullmodem is a special case. The property $\mathsf{G}\,\mathsf{F}\,p \vee \mathsf{F}\,\mathsf{G}\,q$ is not expressible in CTL without using fairness constraints. All three approaches used for that example use a fairness constraint and compute nested fixpoints. The example is included to show that there are cases in which the LTL model checking algorithm that uses an extra automaton is faster than the standard fair CTL algorithm.

We next examine the effects of applying hints. Results for invariant checking are reported in [32]; hence here we only consider properties of other types. Our current implementation only supports hints for properties that translate into terminal automata. Therefore Table 2 has results only for a subset of the experiments shown in Table 1.

The hint used in the two model checking experiments for the *Soap* model is to prevent some of the processors from issuing requests. In the case of the property that fails, it is

Table 1. Comparing model checking approaches for various LTL properties.

experiment	procedure	time (sec)	EXs+ EYs	memory (MB)	peak BDD nodes
Soap (140 latches)	CTL	580	55+45	635	19.4M
$\mathsf{G}\,(p \rightarrow \mathsf{F}\,q)$	$\neg$EF EG fair	639	56+45	644	18.8M
	$\neg$EG$_{\mathsf{fair}}$ **true**	9598	311+45	637	19.3M
Soap	CTL	42	16+45	146	3.4M
$\mathsf{G}\,(p \rightarrow \mathsf{X}\,(q\;\mathsf{R}\;r))$	$\neg$EF fair	8	0+13	43	0.8M
(failing)	$\neg$EF EG fair	260	17+53	570	15.4M
	$\neg$EG$_{\mathsf{fair}}$ **true**	288	32+53	571	15.4M
Soap	CTL	29	4+45	99	2.1M
$\mathsf{G}\,(p \rightarrow \mathsf{X}\,(q\;\mathsf{R}\;r))$	$\neg$EF fair	78	0+45	300	6.0M
(passing)	$\neg$EF EG fair	80	3+45	233	6.0M
	$\neg$EG$_{\mathsf{fair}}$ **true**	77	3+45	233	6.0M
Gcd (45 latches)	CTL	1131	19+0	639	21.2M
$\mathsf{G}\,(p \rightarrow \mathsf{X}\,\mathsf{F}\,q)$	$\neg$EF EG fair	291	19+0	591	15.0M
	$\neg$EG$_{\mathsf{fair}}$ **true**	8831	186+11	659	19.8M
Pcell (61 latches)	CTL	3	47+66	23	120k
$\mathsf{G}\,(p \rightarrow p\;\mathsf{U}\;q)$	$\neg$EF EG fair	4	45+80	25	195k
	$\neg$EG$_{\mathsf{fair}}$ **true**	56	1252+80	73	973k
Palu (99 latches)	CTL	2	5+0	23	228k
$\mathsf{G}\,(p \rightarrow \mathsf{F}\,q)$	$\neg$EF EG fair	3	6+0	25	285k
	$\neg$EG$_{\mathsf{fair}}$ **true**	3	12+0	26	394k
Fpmult (60 latches)	CTL	0.2	5+0	14	12.0k
$\mathsf{G}\,(p \rightarrow \mathsf{X}\,\mathsf{X}\,\mathsf{X}\,q)$	$\neg$EF fair	2.9	0+6	15	53.9k
	$\neg$EF EG fair	0.2	6+0	14	13.6k
	$\neg$EG$_{\mathsf{fair}}$ **true**	0.2	10+0	14	13.7k
NullModem (53 latches)	CTL	1143	28800+388	49	387k
$\mathsf{G}\,\mathsf{F}\,p \vee \mathsf{F}\,\mathsf{G}\,q$	$\neg$EF EG fair	1225	28623+388	42	310k
	$\neg$EG$_{\mathsf{fair}}$ **true**	797	17250+388	91	1120k

Table 2. Effects of guided search. The lines without hints are taken from Table 1.

experiment	procedure	time (sec)	EXs+ EYs	memory (MB)	peak BDD nodes
Soap $\mathsf{G}\,(p \rightarrow \mathsf{X}\,(q\;\mathsf{R}\;r))$	no	7.5	0+13	43	773k
(failing)	yes	1.0	0+13	17	39k
Soap $\mathsf{G}\,(p \rightarrow \mathsf{X}\,(q\;\mathsf{R}\;r))$	no	78	0 + 45	300	6.0M
(passing)	yes	27	0+60	90	2.0M
Fpmult $\mathsf{G}\,(p \rightarrow \mathsf{X}\,\mathsf{X}\,\mathsf{X}\,q)$	no	2.9	0+6	15	53.9k
	yes	1.9	0+15	15	53.6k

indeed possible to generate a counterexample when requests from one processor only are enabled. Considerable speed-up was obtained with a generic hint not specifically targeted at the property. Conversely, the hint that is optimal for the property that fails (only one processor is enabled) is not optimal for the property that passes, but still improves runtime with respect to the standard algorithm. The time to devise the hint was small compared to the time required to formulate the properties. Still, larger test cases than those presented in Table 2 will be required to assess the practical impact of guided search in symbolic model checking.

6 Conclusions and Future Work

In this paper we have presented an efficient algorithm for BDD-based LTL model checking based on guided search and specialized decision procedures for classes of automata. Our algorithm improves on the standard approach in both theory and practice: The selection of the most appropriate decision procedure for an LTL property decreases the asymptotic complexity of the model checking algorithm from quadratic in the size of the state space to linear in many cases, while our preliminary experimental results show that both classification of the automata and the use of hints can have large impacts on the runtime and memory requirements.

Considerable work remains to be done besides the completion of the experimental evaluation of our algorithm. We outline a few of the issues that we plan to explore.

Given an algorithm for CTL model checking with fairness constraints and a tableau construction procedure, a model checker for CTL* is readily available. Also, given the ability to compute least and greatest fixpoints, a μ-calculus model checker can be built. Therefore, our guided search approach can be applied to the logics most commonly used in model checking.

We have seen that LTL model checking may be faster than CTL model checking for the same property, because of the reduction in the number and alternation depth of fixpoints. The opposite may also occur, due to the additional state variables brought by the automaton that may increase the sizes of the BDDs manipulated by the model checker. Direct translation to μ-calculus [17] should therefore be considered as an alternative to composition with the automaton. A better understanding of the relation between different tableau construction procedures is also desirable.

The use of hints is not restricted to BDD-based approaches, but should apply to SAT-based model checking as well [2]. Finally, partial automation of the extraction of hints from the model and the property appears as a worthwhile research goal.

References

1. I. Beer, S. Ben-David, and A. Landver. On-the-fly model checking of RCTL formulas. In A. J. Hu and M. Y. Vardi, editors, *Tenth Conference on Computer Aided Verification (CAV'98)*, pages 184–194. Springer-Verlag, Berlin, 1998. LNCS 1427.
2. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. Unpublished manuscript, October 1998.

3. R. K. Brayton et al. VIS: A system for verification and synthesis. In T. Henzinger and R. Alur, editors, *Eigth Conference on Computer Aided Verification (CAV'96)*, pages 428–432. Springer-Verlag, Rutgers University, 1996. LNCS 1102.
4. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
5. J. R. Burch, E. M. Clarke, and D. E. Long. Representing circuits more efficiently in symbolic model checking. In *Proceedings of the Design Automation Conference*, pages 403–407, San Francisco, CA, June 1991.
6. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, June 1990.
7. H. Cho, G. D. Hachtel, E. Macii, B. Plessier, and F. Somenzi. Algorithms for approximate FSM traversal based on state space decomposition. *IEEE Transactions on Computer-Aided Design*, 15(12):1465–1478, December 1996.
8. E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In D. L. Dill, editor, *Sixth Conference on Computer Aided Verification (CAV'94)*, pages 415–427. Springer-Verlag, Berlin, 1994. LNCS 818.
9. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings Workshop on Logics of Programs*, pages 52–71, Berlin, 1981. Springer-Verlag. LNCS 131.
10. O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines using boolean functional vectors. In L. Claesen, editor, *Proceedings IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, Leuven, Belgium, November 1989.
11. J. Desel and E. Kindler. Proving correctness of distributed algorithms using high-level Petri nets: A case study. In *International Conference on Application of Concurrency to System Design*, Aizu, Japan, March 1998.
12. E. A. Emerson. Temporal and modal logic. In van Leeuwen [33], chapter 16, pages 995–1072.
13. E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of the First Annual Symposium of Logic in Computer Science*, pages 267–278, June 1986.
14. D. Geist and I. Beer. Efficient model checking by automated ordering of transition relation partitions. In D. L. Dill, editor, *Sixth Conference on Computer Aided Verification (CAV'94)*, pages 299–310, Berlin, 1994. Springer-Verlag. LNCS 818.
15. R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification*, pages 3–18. Chapman & Hall, 1995.
16. R. H. Hardin, R. P. Kurshan, S. K. Shukla, and M. Y. Vardi. A new heuristic for bad cycle detection using BDDs. In O. Grumberg, editor, *Ninth Conference on Computer Aided Verification (CAV'97)*, pages 268–278. Springer-Verlag, Berlin, 1997. LNCS 1254.
17. T. A. Henzinger, O. Kupferman, and S. Qadeer. From *pre*-historic to *post*-modern symbolic model checking. In A. J. Hu and M. Y. Vardi, editors, *Tenth Conference on Computer Aided Verification (CAV'98)*, pages 195–206. Springer-Verlag, Berlin, 1998. LNCS 1427.
18. H. Iwashita and T. Nakata. Forward model checking techniques oriented to buggy designs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 400–405, San Jose, CA, November 1997.
19. H. Iwashita, T. Nakata, and F. Hirose. CTL model checking based on forward state traversal. In *Proceedings of the International Conference on Computer-Aided Design*, pages 82–87, San Jose, CA, November 1996.
20. O. Kupferman and M. Y. Vardi. Freedom, weakness, and determinism: From linear-time to branching-time. In *Proc. 13th IEEE Symposium on Logic in Computer Science*, June 1998.

21. O. Kupferman and M. Y. Vardi. Relating linear and branching model checking. In *IFIP Working Conference on Programming Concepts and Methods*, New York, June 1998. Chapman & Hall.
22. R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes*. Princeton University Press, Princeton, NJ, 1994.
23. W. Lee, A. Pardo, J. Jang, G. Hachtel, and F. Somenzi. Tearing based abstraction for CTL model checking. In *Proceedings of the International Conference on Computer-Aided Design*, pages 76–81, San Jose, CA, November 1996.
24. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, New Orleans, January 1985.
25. D. E. Long. *Model Checking, Abstraction, and Compositional Verification*. PhD thesis, Carnegie-Mellon University, July 1993.
26. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, MA, 1994.
27. C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design*. Springer, Berlin, 1998.
28. I.-H. Moon, J.-Y. Jang, G. D. Hachtel, F. Somenzi, C. Pixley, and J. Yuan. Approximate reachability don't cares for CTL model checking. In *Proceedings of the International Conference on Computer-Aided Design*, pages 351–358, San Jose, CA, November 1998.
29. D. E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings of the 3rd IEEE Symposium on Logic in Computer Science*, pages 422–427, Edinburgh, UK, July 1988.
30. A. Pardo and G. D. Hachtel. Incremental CTL model checking using BDD subsetting. In *Proceedings of the Design Automation Conference*, pages 457–462, San Francisco, CA, June 1998.
31. R. K. Ranjan, A. Aziz, R. K. Brayton, B. F. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. Presented at IWLS95, Lake Tahoe, CA., May 1995.
32. K. Ravi. *Adaptive Techniques to Improve State Space Search in Formal Verification*. PhD thesis, University of Colorado, Department of Electrical and Computer Engineering, 1999.
33. J. van Leeuwen, editor. *Handbook of Theoretical Computer Science*. The MIT Press/Elsevier, Amsterdam, 1990.
34. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, UK, June 1986.
35. C. H. Yang and D. L. Dill. Validation with guided search of the state space. In *Proceedings of the Design Automation Conference*, pages 599–604, San Francisco, CA, June 1998.