From *Pre*-historic to *Post*-modern Symbolic Model Checking*

Thomas A. Henzinger Orna Kupferman Shaz Qadeer

Department of EECS, University of California at Berkeley, CA 94720-1770, USA
Email:{tah,orna,shaz}@eecs.berkeley.edu

Abstract. Symbolic model checking, which enables the automatic verification of large systems, proceeds by calculating with expressions that represent state sets. Traditionally, symbolic model-checking tools are based on *backward* state traversal; their basic operation is the function pre, which given a set of states, returns the set of all predecessor states. This is because specifiers usually employ formalisms with future-time modalities, which are naturally evaluated by iterating applications of pre. It has been recently shown experimentally that symbolic model checking can perform significantly better if it is based, instead, on *forward* state traversal; in this case, the basic operation is the function $post$, which given a set of states, returns the set of all successor states. This is because forward state traversal can ensure that only those parts of the state space are explored which are reachable from an initial state and relevant for satisfaction or violation of the specification; that is, errors can be detected as soon as possible.

In this paper, we investigate which specifications can be checked by symbolic forward state traversal. We formulate the problems of symbolic backward and forward model checking by means of two μ-calculi. The pre-μ calculus is based on the pre operation; the $post$-μ calculus, on the $post$ operation. These two μ-calculi induce query logics, which augment fixpoint expressions with a boolean emptiness query. Using query logics, we are able to relate and compare the symbolic backward and forward approaches. In particular, we prove that all ω-regular (linear-time) specifications can be expressed as $post$-μ queries, and therefore checked using symbolic forward state traversal. On the other hand, we show that there are simple branching-time specifications that cannot be checked in this way.

1 Introduction

Today's rapid development of complex and safety-critical systems requires verification methods such as model checking. In model checking [CE81,QS81], we ensure that a system exhibits a desired behavior by executing an algorithm that checks whether a mathematical model of the system satisfies a formal specification that describes the behavior. The algorithmic nature of model checking makes it fully automatic, and thus attractive to practitioners. At the same time, model checking is very sensitive to the size of the mathematical model of the system. Commercial model-checking tools need to cope with the exceedingly large state spaces that are present in real-life examples, making the so-called state-explosion problem perhaps the most challenging issue in computer-aided verification. One of the important developments in this area is the discovery of *symbolic* model-checking methods [BCM+92]. In particular, use of BDDs [Bry86] for model representation has yielded model-checking tools that can handle very large state spaces [CGL93].

Traditional symbolic model-checking tools have been based on *backward* state traversal [McM93,BHSV+96]. They compute with expressions that represent state sets using, in addition to positive boolean operations, the functions pre and $\widetilde{pre}$, which map a set of states to a subset

* This work is supported in part by ONR YIP award N00014-95-1-0520, by NSF CAREER award CCR-9501708, by NSF grant CCR-9504469, by ARO MURI grant DAAH-04-96-1-0341, and by the SRC contract 97-DC-324.041.

of its *predecessor* states. Formally, given a set U of states, the set $pre(U)$ contains the states for which there exists a successor state in U, and the set $\widetilde{pre}(U)$ contains the states all of whose successor states are in U. By evaluating fixpoint expressions over boolean and pre operations, complicated state sets can be calculated. For example, to find the set of states from which a state satisfying a predicate p is reachable, the model checker starts with the set U of states in which p holds, and repeatedly adds to U the set $pre(U)$, until no more states can be added. Formally, the model checker calculates the least fixpoint of the expression $U = p \vee pre(U)$. Symbolic model-checking techniques were first applied to branching-time specifications, and later extended to linear-time specifications, both via translations into fixpoint expressions [BCM+92,CGH94].

As an alternative to symbolic model checking, in *enumerative* model checking states are represented individually. Traditional enumerative model-checking tools check linear-time specifications by *forward* state traversal [Hol97,Dil96]. There, the basic operation is to compute, for a given state, the list of successor states. Forward state traversal has several obvious advantages over backward state traversal. First, for operational system models, successor states are often easier to compute than predecessor states. Second, only the reachable part of the state space is traversed. Third, optimizations such as *on-the-fly* [GPVW95] and *partial-order* [Pel94] methods can be incorporated naturally. For example, in on-the-fly model checking, only those parts of the state space are traversed which are relevant for satisfying (or violating) the given specification.

Some of the advantages of forward state traversal can be easily incorporated into symbolic methods. For example, we may first compute the set of reachable states by symbolic forward state traversal, and then restrict backward state traversal for model checking to the reachable states. This method, however, is unsatisfactory; for example, it cannot find even a short error trace if the set of reachable states cannot be computed. We present a tighter, and more advantageous, integration of forward state traversal with symbolic methods. In *symbolic forward* state traversal, we replace the functions pre and $\widetilde{pre}$ by the functions $post$ and $\widetilde{post}$, respectively, which map a set of states to a subset of its *successor* states. Formally, given a set U of states, the set $post(U)$ contains the states for which there exists a predecessor state in U, and the set $\widetilde{post}(U)$ contains the states all of whose predecessor states are in U. Then, we evaluate fixpoint expressions over boolean and post operations on state sets. It has recently been shown that certain branching-time as well as linear-time specifications, such as response (i.e., $\Box(p \rightarrow \Diamond q)$), can be model checked by symbolic forward state traversal [INH96,IN97]. We attempt a more systematic study of what can and what cannot be model checked in this way. In particular, we show that all ω-regular (linear-time) specifications (which include all LTL specifications) are amenable to a symbolic forward approach, while some CTL (branching-time) specifications are not.

For this purpose, we define *post*-μ, a fixpoint calculus that is based on post operations in the same way in which the traditional μ-calculus, here called pre-μ, is based on pre operations [Koz83]. While pre-μ expressions refer to the future of a given state in a model, $post$-μ expressions refer to its past. Therefore, in stark contrast to the fact that every LTL and CTL specification has an equivalent expression in pre-μ, almost no LTL or CTL specification, including response, has an equivalent expression in $post$-μ. In order to compare pre and post logics, rather, we need to define *query logics*, whose formulas refer to a whole model, not an individual state. Query logics are based on the *emptiness predicate* $\mathcal{E}$. For a specification ϕ, which is true in some states of a model and false in others, the query $\mathcal{E}(\phi)$ is true in a model iff ϕ is false in all states of the model. The query logic $post$-$\mu_\emptyset$ contains all queries of the form $\mathcal{E}(\phi)$ and $\neg\mathcal{E}(\phi)$, for $post$-μ expressions ϕ. On the positive side, we prove that every ω-regular (Büchi) specification has an equivalent query in $post$-$\mu_\emptyset$. As with the translation from Büchi automata to pre-μ expressions [EL86,BC96], the translation from Büchi automata to $post$-$\mu_\emptyset$ queries is linear and involves only fixpoint expressions of alternation depth two. Moreover, we show that every co-Büchi specification has an equivalent query in alternation-free $post$-$\mu_\emptyset$, which can be checked efficiently (in linear time). On the negative side, we prove that there are CTL specifications that are not equivalent to any boolean combination of $post$-$\mu_\emptyset$ queries.

Symbolic forward model checking combines the benefits of symbolic over enumerative state traversal with the benefits of forward over backward state traversal. In [INH96,IN97], the authors present experimental evidence that symbolic forward state traversal can be significantly more efficient than symbolic backward state traversal. Our preliminary experimental results confirm this observation. In addition, we give some theoretical justifications for the symbolic forward approach. We show that unlike enumerative forward model checking (which is traditionally based on depth-first state traversal) and unlike symbolic backward model checking, the symbolic forward approach guarantees *a.s.a.p. error detection*. Intuitively, if a model violates a safety specification, and the shortest error trace has length m, then the breadth-first nature of symbolic forward model checking ensures that the error will be found before any states at a distance greater than m from the initial states are explored.

The remainder of this paper is organized as follows. In Section 2 we define the logics pre-μ and $post$-μ, and the query logics they induce. In Section 3, we translate Büchi automata into equivalent $post$-$\mu_\emptyset$ queries of alternation depth two, and co-Büchi automata into equivalent alternation-free $post$-$\mu_\emptyset$ queries. We also show that the translation guarantees a.s.a.p. error detection for safety specifications. In Section 4, we compare the distinguishing and expressive powers of the various pre, post, and query logics. Finally, in Section 5 we put our results in perspective and report on some experimental evidence for the value of symbolic forward model checking.

2 Definition of Pre and Post Logics

2.1 Pre and post μ-calculi

The μ-calculus is a modal logic augmented with least and greatest fixpoint operators [Koz83]. In this paper, we use the equational form of the propositional μ-calculus, as in [BC96]. The modalities of the μ-calculus relate a set of states to a subset of its predecessor states. Therefore, we refer to the μ-calculus by pre-μ.

The formulas of pre-μ are defined with respect to a set P of propositions and a set V of variables. A *modal expression* is either p, $\neg p$, X, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\exists\bigcirc\varphi$, or $\forall\bigcirc\varphi$, for propositions $p \in P$, variables $X \in V$, and modal expressions φ and ψ. Let I be a finite subset of the set of natural numbers. An *equational block* $B = \langle \lambda, \{X_i = \varphi_i \mid i \in I\}\rangle$ consists of a flag $\lambda \in \{\mu, \nu\}$ and a finite set of equations $X_i = \varphi_i$, where each X_i is a variable, each φ_i is a modal expression, and the variables X_i are pairwise distinct. If $\lambda = \mu$, then B is a *μ-block*; otherwise B is a *ν-block*. For the equational block B, let $vars(B) = \{X_i \mid i \in I\}$ be the set of variables on the left-hand sides of the equations of B. A *block tuple* $\mathcal{B} = \langle B_1, \ldots, B_n\rangle$ is a finite list of equational blocks such that the variable sets $vars(B_j)$, for $1 \leq j \leq n$, are pairwise disjoint. For the block tuple $\mathcal{B}$, let $vars(\mathcal{B}) = \bigcup_{1 \leq j \leq n} vars(B_j)$. For every variable $X \in vars(\mathcal{B})$, let $expand_{\mathcal{B}}(X)$ be the modal expression on the right-hand side of the unique equation in $\mathcal{B}$ whose left-hand side is X. A *pre-μ formula* $\phi = \langle \mathcal{B}, X_0\rangle$ is a pair that consists of a block tuple $\mathcal{B}$ and a variable $X_0 \in vars(\mathcal{B})$. The variable X_0 is called the *root variable* of ϕ. The formula ϕ is a *pre-μ sentence* if every variable that occurs in some modal expression of $\mathcal{B}$ is contained in $vars(\mathcal{B})$.

The semantics of a pre-μ formula is defined with respect to a Kripke structure and a valuation for the variables. A *Kripke structure* is a tuple $K = \langle P, W, R, L\rangle$ that consists of a finite set P of propositions, a finite set W of states, a binary transition relation $R \subseteq W \times W$ total in both the first and second arguments (i.e., for every state $w \in W$, there is a state w' such that $R(w, w')$ and there is a state w'' such that $R(w'', w)$), and a labeling function $L : W \to 2^P$ that assigns to each state a set of propositions. The set P of propositions contains the distinguished proposition $init$; a state $w \in W$ is *initial* if $init \in L(w)$. We define four functions pre, $\widetilde{pre}$, $post$ and $\widetilde{post}$

from 2^W to 2^W as follows. For any set $U \subseteq W$ of states, let

$$\begin{aligned}
pre(U) &= \{w \in W \mid \text{ there exists a state } w' \in U \text{ with } R(w,w')\},\\
\widetilde{pre}(U) &= \{w \in W \mid \text{ for all states } w' \text{ with } R(w,w'), \text{ we have } w' \in U\},\\
post(U) &= \{w \in W \mid \text{ there exists a state } w' \in U \text{ with } R(w',w)\},\\
\widetilde{post}(U) &= \{w \in W \mid \text{ for all states } w' \text{ with } R(w',w), \text{ we have } w' \in U\}.
\end{aligned}$$

A K*-valuation* for a set V of variables is a function $\Gamma : V \to 2^W$ that assigns to each variable a set of states. If Γ and Γ' are K-valuations for V, and $V' \subseteq V$ is a subset of the variables, we write $\Gamma[\Gamma'/V']$ for the K-valuation for V that assigns $\Gamma'(X)$ to each variable $X \in V'$, and $\Gamma(X)$ to each variable $X \in V \backslash V'$.

Given a Kripke structure $K = \langle P, W, R, L\rangle$ and a K-valuation Γ for a set V of variables, every modal expression φ over the propositions P and the variables V defines a set $\varphi^K(\Gamma) \subseteq W$ of states: inductively, $p^K(\Gamma) = \{w \in W \mid p \in L(w)\}$, $(\neg p)^K(\Gamma) = \{w \in W \mid p \notin L(w)\}$, $X^K(\Gamma) = \Gamma(X)$, $(\varphi \vee \psi)^K(\Gamma) = \varphi^K(\Gamma) \cup \psi^K(\Gamma)$, $(\varphi \wedge \psi)^K(\Gamma) = \varphi^K(\Gamma) \cap \psi^K(\Gamma)$, $(\exists\bigcirc\varphi)^K(\Gamma) = pre(\varphi^K(\Gamma))$, and $(\forall\bigcirc\varphi)^K(\Gamma) = \widetilde{pre}(\varphi^K(\Gamma))$. Given K, every block tuple $\mathcal{B} = \langle B_1, \ldots, B_n\rangle$ over P and V defines a function $\mathcal{B}^K$ from the K-valuations for V to the K-valuations for V: inductively, if $n = 0$, then $\mathcal{B}^K(\Gamma) = \Gamma$; if B_1 is a μ-block, then $\mathcal{B}^K(\Gamma)$ is the least fixpoint of the function $F^K_{\mathcal{B},\Gamma}$; if B_1 is a ν-block, then $\mathcal{B}^K(\Gamma)$ is the greatest fixpoint of $F^K_{\mathcal{B},\Gamma}$. The monotonic function $F^K_{\mathcal{B},\Gamma}$ from valuations to valuations is defined by

$$F^K_{\mathcal{B},\Gamma}(\Gamma')(X) \;=\; \begin{cases} expand(X)^K(\langle B_2,\ldots,B_n\rangle^K(\Gamma[\Gamma'/vars(B_1)])) & \text{if } X \in vars(B_1),\\ (\langle B_2,\ldots,B_n\rangle^K(\Gamma[\Gamma'/vars(B_1)]))(X) & \text{otherwise.}\end{cases}$$

Note that for a pre-μ sentence $\phi = \langle \mathcal{B}, X_0\rangle$, the function $\mathcal{B}^K$ is a constant function. Given K, the sentence ϕ defines the set $\phi^K = \mathcal{B}^K(\Gamma)(X_0)$ of states (for any choice of Γ). For a state $w \in W$ and a pre-μ sentence ϕ, we write $w \models_K \phi$ if $w \in \phi^K$. For a Kripke structure K, we write $K \models \phi$, and say that K *satisfies* ϕ, if there is an initial state w of K such that $w \models_K \phi$.[1] The *model-checking problem for* pre-μ is to decide, given a Kripke structure K and a pre-μ sentence ϕ, whether $K \models \phi$.

Given a block tuple $\mathcal{B} = \langle B_1, \ldots, B_n\rangle$, the block B_i *depends* on the block B_j if $i \neq j$ and some variable that occurs in a modal expression of B_i is contained in $vars(B_j)$. The pre-μ sentence $\phi = \langle \mathcal{B}, X_0\rangle$ is *alternation-free* if the dependency relation on the blocks of $\mathcal{B}$ is acyclic (i.e., its transitive closure is asymmetric). The model-checking problem for the alternation-free fragment of pre-μ can be solved in linear time [CS91].

The logic $post$-μ is obtained from the logic pre-μ by replacing the future modal operators $\exists\bigcirc$ and $\forall\bigcirc$ by the past modal operators $\exists\ominus$ and $\forall\ominus$, with the interpretations $(\exists\ominus\varphi)^K(\Gamma) = post(\varphi^K(\Gamma))$ and $(\forall\ominus\varphi)^K(\Gamma) = \widetilde{post}(\varphi^K(\Gamma))$. The semantics of $post$-μ can alternatively be defined as follows. For a Kripke structure $K = \langle P, W, R, L\rangle$, define the Kripke structure $K^{-1} = \langle P, W, R^{-1}, L\rangle$, where $R^{-1}(w, w')$ iff $R(w', w)$. For a $post$-μ sentence ϕ, define ϕ^{-1} to be the pre-μ sentence obtained from ϕ by replacing each occurrence of $\exists\ominus$ and $\forall\ominus$ by $\exists\bigcirc$ and $\forall\bigcirc$, respectively. Then, for every state w of K, we have $w \models_K \phi$ iff $w \models_{K^{-1}} \phi^{-1}$.

2.2 Query logics

We define query logics that are based on pre-μ and $post$-μ. The sentences of pre-μ refer to the future of a given state in a Kripke structure, and the sentences of $post$-μ refer to its past. By contrast, the sentences of query logics, called *queries*, refer to the whole structure and thus enable us to translate between pre and post logics. The query logics are obtained from pre-μ and

[1] Note that we work, for convenience, with the dual of the usual requirement that *all* initial states satisfy a pre-μ sentence.

post-μ by adding a predicate $\mathcal{E}$ on sentences, called the *emptiness predicate*. For a logic $\mathcal{L}$, the query logic $\mathcal{L}_\emptyset$ contains the two queries $\mathcal{E}(\phi)$ and $\neg\mathcal{E}(\phi)$ for each sentence ϕ of $\mathcal{L}$. The query logic $\mathcal{L}_\mathcal{E}$ is richer and its queries are constructed inductively as follows:

– $\mathcal{E}(\phi)$, where ϕ is a formula of $\mathcal{L}$,
– $\neg\theta_1$ and $\theta_1 \vee \theta_2$, where θ_1 and θ_2 are queries of $\mathcal{L}_\mathcal{E}$.

The satisfaction relation $\models$ for queries on a Kripke structure K is inductively defined as follows:

– $K \models \mathcal{E}(\phi)$ iff for all states s of K, we have $s \not\models \phi$,
– $K \models \neg\theta_1$ iff $K \not\models \theta_1$, and $K \models \theta_1 \vee \theta_2$ iff $K \models \theta_1$ or $K \models \theta_2$.

While our motivation for query logics is theoretical, for the purpose of comparing pre and post logics, query logics are also practical. This is because once the state set ϕ^K has been computed (either explicitly or implicitly, using BDDs), the evaluation of the query $\mathcal{E}(\phi)$ requires constant time. Therefore, checking a query in pre-$\mu_\mathcal{E}$ or *post-$\mu_\mathcal{E}$* is no harder than model checking pre-μ or *post-μ*, respectively.

2.3 Equivalences on Kripke structures induced by pre and post logics

Let $K = \langle P, W, R, L\rangle$ and $K' = \langle P, W', R', L'\rangle$ be two Kripke structures with the same set of propositions. A relation $\beta \subseteq W \times W'$ is a *pre-bisimilarity relation* if for all states w and w', we have that $\beta(w, w')$ implies (1) $L(w) = L'(w')$, (2) for every state v with $R(w, v)$, there is a state v' with $R'(w', v')$ and $\beta(v, v')$, and (3) for every state v' with $R'(w', v')$, there is a state v with $R(w, v)$ and $\beta(v, v')$. Note that, in particular, $\beta(w, w')$ implies that either both w and w' are initial, or neither of them is initial. The pre-bisimilarity relation β is a *pre-bisimulation* between K and K' if for all states $w \in W$, there is a state $w' \in W'$ such that $\beta(w, w')$, and for all states $w' \in W'$, there is a state $w \in W$ such that $\beta(w, w')$. The pre-bisimilarity relation β is an *init-pre-bisimulation* between K and K' if for all initial states $w \in W$, there is an initial state $w' \in W'$ such that $\beta(w, w')$, and for all initial states $w' \in W'$, there is an initial state $w \in W$ such that $\beta(w, w')$. The relation $\beta \subseteq W \times W'$ is a *post-bisimulation* (resp. *init-post-bisimulation*) between K and K' if β is a pre-bisimulation (resp. init-pre-bisimulation) between K^{-1} and K'^{-1}. The following is an easy extension of a well-known result for pre-μ [BCG88].

Proposition 1. *Let K and K' be two Kripke structures.*

– *There is an init-pre-bisimulation (resp. init-post-bisimulation) between K and K' iff for all sentences ϕ of pre-μ (resp. post-μ), we have $K \models \phi$ iff $K' \models \phi$.*
– *The following three statements are equivalent:*
  (1) *There is a pre-bisimulation (resp. post-bisimulation) between K and K'.*
  (2) *For all queries θ of pre-$\mu_\emptyset$ (resp. post-$\mu_\emptyset$), we have $K \models \theta$ iff $K' \models \theta$.*
  (3) *For all queries θ of pre-$\mu_\mathcal{E}$ (resp. post-$\mu_\mathcal{E}$), we have $K \models \theta$ iff $K' \models \theta$.*

3 Intersection of Pre and Post Logics

Of particular interest is the intersection of the query logics pre-$\mu_\mathcal{E}$ and *post-$\mu_\mathcal{E}$*. It contains the queries that can be specified in both pre-$\mu_\mathcal{E}$, which often is more convenient for specifiers, and in *post-$\mu_\mathcal{E}$*, which often is more efficient for symbolic model checking. In this section we show that essentially all linear properties lie in this intersection. On the other hand, there are simple branching properties that do not lie in the intersection.

3.1 In

Consider a Kripke structure $K = \langle P, W, R, L\rangle$. An *observation* of K is a subset of the propositions P. An *error trace* of K is a finite or infinite sequence of observations. A *linear property* of K is a set of error traces.[2] Many useful linear properties, namely, the ω-regular linear properties, can be specified by finite automata. A *finite automaton* $A = \langle P, S, S_0, S_F, r, \ell\rangle$ consists of a finite set P of propositions, a finite set S of states, a set $S_0 \subseteq S$ of initial states, a set $S_F \subseteq S$ of accepting states, a binary transition relation $r \subseteq S \times S$, and a labeling function $\ell : S \to 2^P$ that assigns to each state a set of propositions. The following definitions regarding paths apply equally to Kripke structures and automata. A *path* $\pi = u_0, u_1, \ldots$ of K (resp. A) is a finite or infinite sequence of states such that for all $i \geq 0$, we have $R(u_i, u_{i+1})$ (resp. $r(u_i, u_{i+1})$). The path π is *initialized* if u_0 is an initial state. By $\mathit{Inf}(\pi)$ we denote the set of states that appear in π infinitely often. The labeling functions L and ℓ are lifted from states to paths in the obvious way.

With each finite automaton A we associate a sentence $\exists A$ that is interpreted over a Kripke structure K with the same propositions as A. The *model-checking problem for automata* is to decide, given K and A, whether $K \models \exists A$. We define $K \models \exists A$ if there exist an initialized path π_1 of K and an *accepting* initialized path π_2 of A such that $L(\pi_1) = \ell(\pi_2)$; such an observation sequence $L(\pi_1)$ is called an *error trace of K with respect to A*. Which paths of A are accepting depends on the interpretation we place on the automaton A. We consider here three different interpretations: safety automata, Büchi automata, and co-Büchi automata. For each interpretation we reduce the model-checking problem for automata to the model-checking problem for *post*-$\mu_\emptyset$, by translating automata into equivalent *post*-$\mu_\emptyset$ queries. The *post*-$\mu_\emptyset$ query θ is *equivalent* to the automaton A if for every Kripke structure K, we have $K \models \exists A$ iff $K \models \theta$.

In all translations, we will make use of the following. With each state s of the automaton A, we associate two variables, X_s and X'_s. In addition, we use the two variables X_F and X'_F. For each state s of A, let γ_s be a variable-free and modality-free expression that characterizes states locally, namely, $\gamma_s = \bigwedge_{p\in\ell(s)} p \wedge \bigwedge_{p\notin\ell(s)} \neg p$. Now, let B_A be the following μ-block, which consists of $|S| + 1$ equations, with $\mathit{vars}(B_A) = \{X_s \mid s \in S\} \cup \{X_F\}$:

$$X_s = \begin{cases} \gamma_s \wedge (\mathit{init} \vee \bigvee_{t\in pre(s)} \exists\odot X_t) & \text{if } s \in S_0, \\ \gamma_s \wedge \bigvee_{t\in pre(s)} \exists\odot X_t & \text{if } s \notin S_0, \end{cases}$$
$$X_F = \bigvee_{f\in S_F} X_f.$$

Note that the size of B_A is linear in the size of A.

Safety automata A *safety property* of a Kripke structure K is a set of *finite* error traces. The regular safety properties can be specified by safety automata. A *safety automaton* is a finite automaton A such that a path π of A is *accepting* if π is a finite path and its last state is an accepting state of A. It is not difficult to see that the safety automaton A is equivalent to the *post*-$\mu_\emptyset$ query $\theta_A = \neg\mathcal{E}(\langle\langle B_A\rangle, X_F\rangle)$.

If a finite error trace exists, during model checking, we would like to find it as soon as possible. By evaluating the query θ_A as follows (in the standard way), this can indeed be guaranteed. The evaluation of the μ-block B_A over a Kripke structure K proceeds in iterations. Let $X^K(i) \subseteq W$ denote the value of variable $X \in \mathit{vars}(B_A)$ after the *i*-th iteration, and let $\Gamma^K(i)$ denote the K-valuation that assigns to each variable in $X \in \mathit{vars}(B_A)$ the value $X^K(i)$. Initially, $X^K(0) = \emptyset$ for all $X \in \mathit{vars}(B_A)$. In all subsequent iterations, the value of each variable $X \in \mathit{vars}(B_A)$ is updated according to the equation $X^K(i+1) = \mathit{expand}(X)^K(\Gamma^K(i))$. Since the modal expressions in B_A are monotonic, once $X_F^K(m) \neq \emptyset$ for some m, we know that

[2] Recall that we work, for convenience, in a setting that is dual to the one that considers linear properties to consist of all *non*-error traces.

$X_F^K(n) \neq \emptyset$ for all $n > m$. Hence, we can detect that $K \models \theta_A$ as soon as $X_F^K(m)$ is nonempty. The following theorem guarantees that if there is an error trace of length m, then we will find it in m iterations. In other words, using symbolic forward state traversal, we will explore only states up to distance m from initial states.

Theorem 1. *For every safety automaton A, an equivalent alternation-free post-$\mu_\emptyset$ query θ_A can be constructed in linear time. Further, for every Kripke structure K, if the shortest error trace in K with respect to A has length m, then $X_F^K(m) \neq \emptyset$, where X_F is the root variable of θ_A.*

Büchi automata Safety automata cannot specify infinite error traces. For that, we use Büchi automata. A *Büchi automaton* A is a finite automaton such that a path π of A is *accepting* if $\mathit{Inf}(\pi) \cap S_F \neq \emptyset$; that is, some accepting state of A occurs infinitely often in π. It is well-known [EL86,Dam94,BC96] that for every Büchi automaton A, there exists a pre-$\mu_\emptyset$ query ϑ_A such that for every Kripke structure K, we have $K \models \exists A$ iff $K \models \vartheta_A$. We now show that there exists also a *post*-$\mu_\emptyset$ query θ_A with the same property, thereby proving that the model-checking problem for Büchi automata lies in the intersection of pre-$\mu_\emptyset$ and *post*-$\mu_\emptyset$. We define two equational blocks: a ν-block B_1 and a μ-block B_2. The block B_1 contains the following $|S_F| + 1$ equations, with $vars(B_1) = \{X'_f \mid f \in S_F\} \cup \{X'_F\}$:

$$\begin{aligned} X'_f &= X_f \wedge \textstyle\bigvee_{t \in pre(f)} \exists\odot X'_t, \\ X'_F &= \textstyle\bigvee_{f \in S_F} X'_f. \end{aligned}$$

The block B_2 contains an equation for each state $s \in S \backslash S_F$, defined by

$$X'_s = \gamma_s \wedge \textstyle\bigvee_{t \in pre(s)} \exists\odot X'_t.$$

Then, $\theta_A = \neg\mathcal{E}(\langle\langle B_1, B_2, B_A\rangle, X'_F\rangle)$. Notice that, as with pre-$\mu_\emptyset$ [BC96], the translation is linear in the size of the Büchi automaton. Also, the equational blocks B_1 and B_2 depend on each other and the alternation depth of θ_A is two. Since Büchi automata are expressively equivalent to the ω-regular languages, the query logic *post*-$\mu_\emptyset$ can specify all ω-regular properties.

Theorem 2. *For every Büchi automaton, an equivalent post-$\mu_\emptyset$ query of alternation depth two can be constructed in linear time.*

In particular, since all sentences of the linear temporal logic LTL can be translated to Büchi automata [VW94], Theorem 2, together with [EL86], implies that all LTL sentences lie in the intersection pre-$\mu_\emptyset \cap$ *post*-$\mu_\emptyset$. Hence, LTL model checking can proceed by symbolic forward state traversal. Since the translation from LTL to Büchi automata involves an exponential blow-up, the translation from LTL to *post*-$\mu_\emptyset$ is also exponential.

Co-Büchi automata Recall that the translation from Theorem 2 results in formulas of alternation depth two. It has been recently argued [KV98] that a linear property given by a co-Büchi automaton can be translated into an *alternation-free* pre-$\mu_\emptyset$ query.[3] Consequently, the model checking of linear properties that are specified by co-Büchi automata requires time that is only linear in the size of the Kripke structure. We now show that every co-Büchi automaton A can also be translated into an equivalent alternation-free *post*-$\mu_\emptyset$ query θ_A, thereby proving that the model-checking problem for co-Büchi automata lies in the intersection of alternation-free pre-$\mu_\emptyset$ and alternation-free *post*-$\mu_\emptyset$. A *co-Büchi automaton* A is a finite automaton such that a path π of

[3] The results in [KV98] refer to sentences of the form $\forall A$, for deterministic Büchi automata A. Since an ω-regular language can be specified by a deterministic Büchi automaton iff its complement can be specified by a co-Büchi automaton, the corresponding result for $\exists A$, for co-Büchi automata A, follows by duality.

A is *accepting* if $Inf(\pi) \subseteq S_F$; that is, all the non-accepting states of A occur in π only finitely often. We define an equational ν-block B_3 that contains the following $|S_F| + 1$ equations, with $vars(B_3) = \{X'_f \mid f \in S_F\} \cup \{X'_F\}$:

$$\begin{aligned} X'_f &= X_f \wedge \bigvee_{t \in pre(f) \cap S_F} \exists \ominus X'_t, \\ X'_F &= \bigvee_{f \in S_F} X'_f. \end{aligned}$$

Then, $\theta_A = \neg\mathcal{E}(\langle\langle B_3, B_A\rangle, X'_F\rangle)$. Notice that θ_A is alternation-free and linear in the size of A.

Theorem 3. *For every co-Büchi automaton, an equivalent alternation-free post-$\mu_\emptyset$ query can be constructed in linear time.*

3.2 Out

We now show that there exist branching temporal-logic specifications that cannot be model checked by evaluating *post-*$\mu_\mathcal{E}$ queries. A *post-*$\mu_\mathcal{E}$ query θ is *equivalent* to a *pre-*μ sentence ϕ if for every Kripke structure K, we have $K \models \phi$ iff $K \models \theta$. Consider the *pre-*μ sentence $\phi_1 = \langle\langle\langle \nu, \{X = \exists\bigcirc p \wedge \exists\bigcirc X\}\rangle\rangle, X\rangle$, which is equivalent to the CTL sentence $\exists\Box\exists\bigcirc p$, and consider the Kripke structures K_1 and K'_1 appearing in Figure 1. It is easy to see that

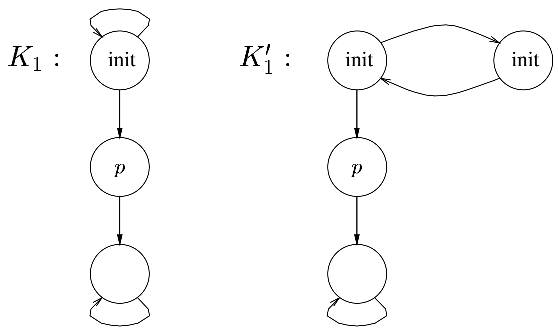


Fig. 1. K_1 and K'_1 are post-bisimilar but not pre-bisimilar.

there is a post-bisimulation between K_1 and K'_1. Hence, by Proposition 1, no *post-*$\mu_\mathcal{E}$ query can distinguish between them. On the other hand, while the structure K_1 satisfies ϕ_1, the structure K'_1 does not satisfy ϕ_1. Using a similar argument, it can be shown that the *pre-*μ sentence that is equivalent to the CTL sentence $\exists\Diamond(r \wedge \exists\Diamond p \wedge \exists\Diamond q)$ can distinguish between two structures that have a post-bisimulation between them, implying there is no equivalent *post-*$\mu_\mathcal{E}$ query. Interestingly, the *pre-*μ sentence $\langle\langle B_1, B_2\rangle, X_1\rangle$ with $B_1 = \langle \nu, \{X_1 = X_2 \wedge \exists\bigcirc X_1\}\rangle$ and $B_2 = \langle \mu, \{X_2 = p \vee \exists\bigcirc X_2\}\rangle$, which is equivalent to the CTL sentence $\exists\Box\exists\Diamond p$, and which is not equivalent to any LTL sentence [CD88], does have an equivalent query in *post-*$\mu_\emptyset$. The query is $\neg\mathcal{E}(\langle\langle B_3, B_4\rangle, X_1\rangle)$, with $B_3 = \langle \nu, \{X_1 = p \wedge X_2, X_2 = X_3 \wedge \exists\ominus X_2\}\rangle$ and $B_4 = \langle \mu, \{X_3 = \mathit{init} \vee \exists\ominus X_3\}\rangle$.

Proposition 2. *There exist pre-μ sentences (in fact, CTL sentences) that have no equivalent post-$\mu_\mathcal{E}$ queries.*

4 Hierarchy of Pre and Post Logics

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two logics whose sentences are interpreted over Kripke structures. The logic $\mathcal{L}_2$ is *as expressive as* the logic $\mathcal{L}_1$ if for every sentence ϕ_1 of $\mathcal{L}_1$, there is a sentence $\phi_2 \in \mathcal{L}_2$

such that for every Kripke structure K, we have $K \models \phi_1$ iff $K \models \phi_2$. The logic $\mathcal{L}_2$ is *more expressive than* $\mathcal{L}_1$ if $\mathcal{L}_2$ is as expressive as $\mathcal{L}_1$ but $\mathcal{L}_1$ is not as expressive as $\mathcal{L}_2$. The logic $\mathcal{L}_2$ is *as distinguishing as* the logic $\mathcal{L}_1$ if for all Kripke structures K and K', if there is a sentence ϕ_1 of $\mathcal{L}_1$ such that $K \models \phi_1$ but $K' \not\models \phi_1$, then there is a sentence ϕ_2 of $\mathcal{L}_2$ such that $K \models \phi_2$ but $K' \not\models \phi_2$. Finally, the logic $\mathcal{L}_2$ is *more distinguishing than* $\mathcal{L}_1$ if $\mathcal{L}_2$ is as distinguishing as $\mathcal{L}_1$ but $\mathcal{L}_1$ is not as distinguishing as $\mathcal{L}_2$. In this section, we study the distinguishing and the expressive powers of *pre*-μ and *post*-μ and the query logics they induce. For this purpose, the sentences of query logics are the queries.

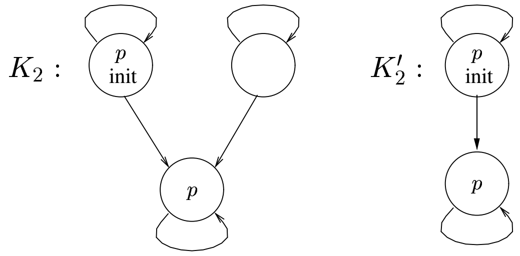
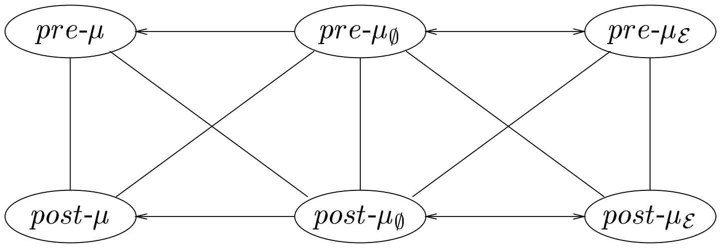


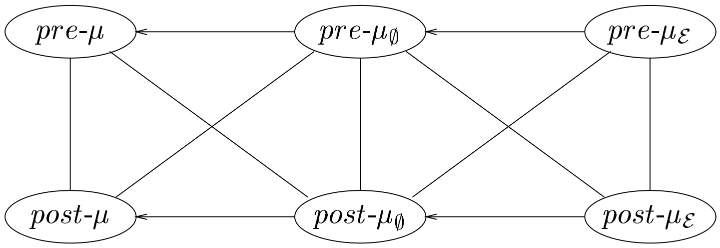
Fig. 2. K_2 and K'_2 are init-post-bisimilar but not post-bisimilar

Proposition 3. *The distinguishing powers of pre and post logics are summarized in the figure below. An arrow from logic $\mathcal{L}_1$ to logic $\mathcal{L}_2$ indicates that $\mathcal{L}_1$ is as distinguishing as $\mathcal{L}_2$. A line without arrow indicates incomparability.*



Proof. Proposition 1 implies that the distinguishing powers of *pre*-$\mu_\emptyset$ and *pre*-$\mu_\mathcal{E}$ coincide, and similarly for *post*-μ. In order to prove the incomparability results, we show that the four relations init-pre-bisimulation, init-post-bisimulation, pre-bisimulation, and post-bisimulation are all distinct. Recall that there may be states in a Kripke structure that are not reachable from an initial state, as there may be states from which no initial state is reachable. Consider the Kripke structures K_2 and K'_2 appearing in Figure 2. There is an init-pre-bisimulation and an init-post-bisimulation between K_2 and K'_2, but no pre-bisimulation or post-bisimulation. Hence, (post) pre-bisimulation is more distinguishing than (init-post) init-pre-bisimulation. Now consider the Kripke structures K_1 and K'_1 appearing in Figure 1. There is a post-bisimulation and an init-post-bisimulation between K_1 and K'_1, but no pre-bisimulation or init-pre-bisimulation. Also, there is a pre-bisimulation and an init-pre-bisimulation between K_1^{-1} and ${K'}_1^{-1}$ but no post-bisimulation or init-post-bisimulation. Hence, pre-bisimulation and post-bisimulation as well as init-pre-bisimulation and init-post-bisimulation are incomparable. □

Proposition 4. *The expressive powers of pre and post logics are summarized in the figure below. An arrow from logic $\mathcal{L}_1$ to logic $\mathcal{L}_2$ indicates that $\mathcal{L}_1$ is as expressive as $\mathcal{L}_2$. A line without arrow indicates incomparability.*



Proof. It is easy to see that if a logic $\mathcal{L}_2$ is not as distinguishing as a logic $\mathcal{L}_1$, then $\mathcal{L}_2$ is not as expressive as $\mathcal{L}_1$. Therefore, most of our expressiveness results follow from the corresponding results about distinguishability. In addition, as a Kripke structure K satisfies a sentence ϕ iff K satisfies the query $\neg\mathcal{E}(init \wedge \phi)$, the query logics $pre\text{-}\mu_\emptyset$ and $post\text{-}\mu_\emptyset$ are more expressive than $pre\text{-}\mu$ and $post\text{-}\mu$, respectively. In order to prove the advantage of the full query logics $pre\text{-}\mu_\mathcal{E}$ and $post\text{-}\mu_\mathcal{E}$ over its subsets $pre\text{-}\mu_\emptyset$ and $post\text{-}\mu_\emptyset$, it is easy to see that no query of the query logics $pre\text{-}\mu_\emptyset$ and $post\text{-}\mu_\emptyset$ is equivalent to the query $\mathcal{E}(p) \vee \mathcal{E}(q)$. $\square$

5 Discussion and Experimental Results

5.1 Intersection of pre and post logics

While previous works presented symbolic forward state-traversal procedures for model checking some isolated linear and branching properties [INH96,IN97], we attempted to study more systematically the class of properties that can be model checked using both symbolic forward and backward state traversal. In particular, we showed that all ω-regular linear properties (which includes all properties expressible in LTL) fall into this class, while some simple branching properties (expressible in CTL) do not. Furthermore, every query that can be specified in both $pre\text{-}\mu_\emptyset$ and $post\text{-}\mu_\emptyset$ cannot distinguish between structures that are both pre-bisimilar and post-bisimilar. Yet the exact characterization of the intersection $pre\text{-}\mu_\emptyset \cap post\text{-}\mu_\emptyset$ remains open. In [GK94], the authors identified a set of temporal-logic sentences called equi-linear. In particular, a $pre\text{-}\mu$ sentence is *equi-linear* if it cannot distinguish between two Kripke structures with the same language (i.e., observation sequences that correspond to initialized paths). Clearly, all LTL sentences are equi-linear. However, some CTL sentences that have no equivalent LTL sentence are also equi-linear. For example, it is shown in [GK94] that while the CTL sentence $\exists\Box\exists\bigcirc p$ is not equi-linear, the CTL sentence $\exists\Box\exists\Diamond p$ is equi-linear. Motivated by the examples from Section 3.2, we conjecture that equi-linearity precisely characterizes the properties that can be model checked using both symbolic forward and backward state traversal. Formally, we conjecture that a $pre\text{-}\mu$ sentence is equi-linear iff there is an equivalent $post\text{-}\mu_\emptyset$ query.

5.2 Union of pre and post logics

In this paper, we primarily think of $post\text{-}\mu_\mathcal{E}$ as a language for describing symbolic model-checking procedures for temporal-logic specifications. Furthermore, we have focused on specification languages that contain only future temporal operators. Since LTL with past temporal operators is no more expressive than LTL without past operators [LPZ85], every LTL+past sentence can also be translated into an equivalent $post\text{-}\mu_\emptyset$ query. In addition, $post\text{-}\mu$ also permits the easy evaluation of branching past temporal operators that cannot be evaluated using $pre\text{-}\mu$. For example, the sentence $\forall\Box(grant \rightarrow (\neg init)\forall\widetilde{\mathcal{W}}req)$, where $\widetilde{\mathcal{W}}$ is a past version of the "weak-until" operator [MP92], specifies that grants are given only upon request. Assuming a *branching* interpretation for past temporal operators [KP95], this sentence has an equivalent $post\text{-}\mu_\emptyset$ query, but no equivalent $pre\text{-}\mu_\mathcal{E}$ query; that is, it can be model checked by symbolic forward state traversal but not by symbolic backward state traversal.

While the intersection $pre\text{-}\mu_{\mathcal{E}} \cap post\text{-}\mu_{\mathcal{E}}$ identifies the queries that can be model checked by both symbolic forward and backward state traversal, it is the "union" $(pre\text{-}\mu \cup post\text{-}\mu)_{\mathcal{E}}$[4] that identifies the queries that can be model checked at all symbolically, by mixed forward and backward state traversal.[5] Furthermore, it is the alternation-free fragment of $(pre\text{-}\mu \cup post\text{-}\mu)_{\mathcal{E}}$ that identifies the queries that can be model checked *efficiently*. Thus it is also of interest to ask which temporal logics can be translated into the (alternation-free) union of pre and post query logics. Such temporal logics can have both future and past temporal operators. In particular, it is easy to see that every CTL+past sentence (under the branching interpretation for past) has an equivalent query in the alternation-free fragment of $(pre\text{-}\mu \cup post\text{-}\mu)_{\emptyset}$.

5.3 Experimental results

In our experiments, we performed BDD-based symbolic model checking on a parameterized sliding-window protocol for the reliable transmission of packets over an unreliable channel. The parameter to the protocol is $WINSIZE$, the number of outstanding unacknowledged messages at the sender end. In the protocol, the messages are modeled as boolean values. We checked whether all computations of the protocol satisfy the partial specification ϕ, which states that if the produced message *msgP* toggles infinitely often at the sender end, then so does the consumed message *msgC* at the receiver end. Formally, the specification ϕ is given by the LTL sentence $\Box\Diamond(msgP \leftrightarrow \bigcirc\neg msgP) \rightarrow \Box\Diamond(msgC \leftrightarrow \bigcirc\neg msgC)$. We note that this sentence cannot be handled by the methods presented in [INH96,IN97].

In the table below we list the running times (in seconds) for different values of $WINSIZE$ for checking ϕ using VIS [BHSV+96] for both symbolic forward and backward state traversal. The quantity within the parentheses is the number of boolean variables used to encode the state space of the protocol. It is folk wisdom in symbolic model checking that using don't-care minimization based on unreachable states can dramatically improve the running times. So we also applied first symbolic forward state traversal to compute the set of reachable states and then symbolic backward state traversal for model checking, using the unreachable states as don't cares. These results are shown in the last column. A dash indicates an unsuccessful verification attempt. In the future, we hope to compare our approach also against *enumerative* forward state-traversal methods for LTL model checking.

WINSIZE	*Forward*	*Backward*	*Reach-optimized backward*
2 (30)	18	222	91
3 (45)	300	4584	-
4 (50)	5231	-	-

Acknowledgments

We thank Rajeev Alur, Bob Brayton, Ed Clarke, Allen Emerson, and Orna Grumberg for helpful discussions, and Carl Pixley for drawing the authors' attention to [INH96].

References

[BC96] G. Bhat and R. Cleavland. Efficient model checking via the equational μ-calculus. In *Proc. 11th IEEE Symposium on Logic in Computer Science*, pp. 304–312, 1996.

[4] By the *union* $pre\text{-}\mu \cup post\text{-}\mu$ we refer to the logic with all four modal operators $\exists\bigcirc$, $\forall\bigcirc$, $\exists\ominus$, and $\forall\ominus$. It has, of course, strictly more sentences than the union of the sets of $pre\text{-}\mu$ and $post\text{-}\mu$ sentences.

[5] In fact, not only can model checking algorithms be extended from $pre\text{-}\mu$ to $(pre\text{-}\mu \cup post\text{-}\mu)_{\mathcal{E}}$ without extra cost, the satisfiability problem for the union is also no harder than the satisfiability problem for either $pre\text{-}\mu$ or $post\text{-}\mu$ [Var98].

[BCG88] M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BHSV+96] R.K. Brayton, G.D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, T. Kukimoto, A. Pardo, S. Qadeer, R.K. Ranjan, S. Sarwary, T.R. Shiple, G. Swamy, and T. Villa. VIS: a system for verification and synthesis. In *CAV 96: Computer Aided Verification*, LNCS 1102, pp. 428–432, Springer, 1996.

[Bry86] R.E. Bryant. Graph-based algorithms for boolean-function manipulation. *IEEE Trans. on Computers*, C-35(8), 1986.

[CD88] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, LNCS 354, pp. 428–437, Springer, 1988.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, LNCS 131, pp. 52–71, Springer, 1981.

[CGH94] E.M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In *CAV 94: Computer Aided Verification*, LNCS 818, pp. 415 – 427, Springer, 1994.

[CGL93] E.M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In *Decade of Concurrency – Reflections and Perspectives (Proc. REX School)*, LNCS 803, pp. 124–175, Springer, 1993.

[CS91] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. In *CAV 91: Computer Aided Verification*, LNCS 575, pp. 48–58, Springer, 1991.

[Dam94] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126:77–96, 1994.

[Dil96] David L. Dill. The Murϕ Verification System. In *CAV 96: Computer Aided Verification*, LNCS 1102, pp. 390–393, Springer, 1996.

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. 1st Symposium on Logic in Computer Science*, pp. 267–278, 1986.

[GK94] O. Grumberg and R.P. Kurshan. How linear can branching-time be. In *Proc. 1st International Conference on Temporal Logic*, LNAI 827, pp. 180–194, Springer, 1994.

[GPVW95] 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*, pp. 3–18, Chapman, 1995.

[Hol97] G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.

[IN97] H. Iwashita and T. Nakata. Forward model checking techniques oriented to buggy designs. In *Proc. IEEE/ACM International Conference on Computer Aided Design*, pp. 400–404, 1997.

[INH96] H. Iwashita, T. Nakata, and F. Hirose. CTL model checking based on forward state traversal. In *Proc. IEEE/ACM International Conference on Computer Aided Design*, pp. 82–87, 1996.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KP95] O. Kupferman and A. Pnueli. Once and for all. In *Proc. 10th IEEE Symposium on Logic in Computer Science*, pp. 25–35, 1995.

[KV98] 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*, 1998.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, LNCS 193, pp. 196–218, Springer, 1985.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[Pel94] D. Peled. Combining partial order reductions with on-the-fly model-checking. In *CAV 94: Computer Aided Verification*, LNCS 818, pp. 377–390, Springer, 1994.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, LNCS 137, pp. 337–351, Springer, 1981.

[Var98] M.Y. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th International Coll. on Automata, Languages, and Programming*, LNCS, Springer, 1998.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.