# The Birth of Model Checking*

Edmund M. Clarke

emc@cs.cmu.edu
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA

"When the time is ripe for certain things, these things appear in different places in the manner of violets coming to light in early spring." (Wolfgang Bolyai to his son Johann in urging him to claim the invention of non-Euclidean geometry without delay [Vit88])

## 1 Model Checking

Model Checking did not arise in a historical vacuum. There was an important problem that needed to be solved, namely Concurrent Program Verification. Concurrency errors are particularly difficult to find by program testing, since they are often hard to reproduce. Most of the formal research on this topic involved constructing proofs by hand using a Floyd-Hoare style logic. Probably, the best known formal system was the one proposed by Owicki and Gries [OG76] for reasoning about Conditional Critical Regions. Although I had written my thesis on the meta-theory of Hoare Logic [Cla77a,Cla77b,Cla78,Cla79a,Cla79c,Cla80] and was very familiar with the Owick-Gries proof methodology, I was quite skeptical about the scalability of hand constructed proofs. There had been some practical research on state exploration methods for communication protocols by Gregor Bochmann and others, but it was largely ignored by the "Formal Verification Community". Also, in the late 1970's, Pnueli [Pnu77] and Owicki and Lamport [OL82] had proposed the use of Temporal Logic for specifying concurrent programs. Although they still advocated hand constructed proofs, their work demonstrated convincingly that Temporal Logic was ideal for expressing concepts like mutual exclusion, absence of deadlock, and absence of starvation.

Allen Emerson and I combined the state-exploration approach with Temporal Logic in an efficient manner and showed that the result could be used to solve non-trivial problems. Here is a quote from our original 1981 paper [CE81]:

* This research was sponsored by the National Science Foundation under grant nos. CNS- 0411152, CCF-0429120, CCR-0121547, and CCR-0098072, the US Army Research Office under grant no. DAAD19-01-1-0485, and the Office of Naval Research under grant no. N00014-01-1-0796. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

"The task of proof construction is in general quite tedious and a good deal of ingenuity may be required to organize the proof in a manageable fashion.We argue that proof construction is unnecessary in the case of finite state concurrent systems and can be replaced by a model-theoretic approach which will mechanically determine if the system meets a specification expressed in propositional temporal logic. The global state graph of the concurrent systems can be viewed as a finite Kripke structure and an efficient algorithm can be given to determine whether a structure is a model of a particular formula (i.e. to determine if the program meets its specification)."

### 1.1 What is Model Checking?

The Model Checking problem is easy to state:

> Let $M$ be a Kripke structure (i.e., state-transition graph). Let $f$ be a formula of temporal logic (i.e., the specification). Find all states $s$ of $M$ such that $M, s \models f$ .

We used the term *Model Checking* because we wanted to determine if the temporal formula $f$ was true in the Kripke structure $M$, i.e., whether the structure $M$ was a *model* for the formula $f$. Some people believe erroneously that the use of the term "model" refers to the dictionary meaning of this word (e.g., a miniature representation of something or a pattern of something to be made) and indicates that we are dealing with an abstraction of the actual system under study.

Emerson and I gave a polynomial algorithm for solving the Model Checking Problem for the logic CTL. The figure below shows the structure of a typical Model Checking system. A preprocessor extracts a state transition graph from a program or circuit. The Model Checking engine takes the state transition graph and a temporal formula and determines whether the formula is true or not (Figure 1).

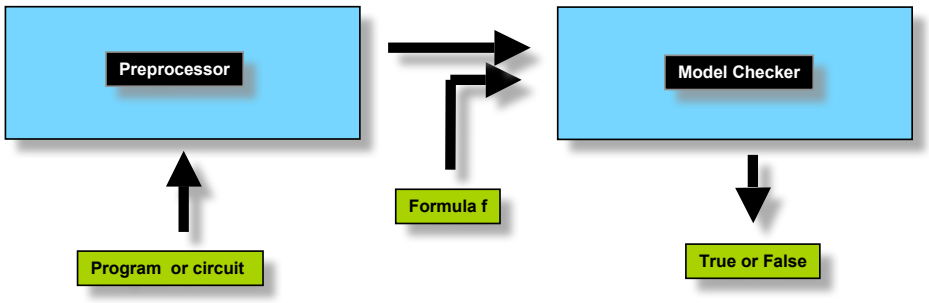


**Fig. 1.** Model Checker Structure

### 1.2 Advantages of Model Checking

Model Checking has a number of advantages compared to other verification techniques such as automated theorem proving or proof checking. A partial list of some of these advantages is given below:

- No proofs! The user of a Model Checker does not need to construct a correctness proof. In principle, all that is necessary is for the user to enter a description of the circuit or program to be verified and the specification to be checked and press the "return" key. The checking process is automatic.
- Fast. In practice, Model checking is fast compared to other rigorous methods such as the use of a proof checker, which may require months of the user's time working in interactive mode.
- Diagnostic counterexamples. If the specification is not satisfied, the Model Checker will produce a counterexample execution trace that shows why the specification does not hold (Figure 2). It is impossible to overestimate the importance of the counterexample feature. The counterexamples are invaluable in debugging complex systems. Some people use Model Checking just for this feature.
- No problem with partial specifications. It is unnecessary to completely specify the program or circuit before beginning to Model Check properties. Thus, Model Checking can be used during the design of a complex system. The user does not have to wait until the design phase is complete.
- Temporal Logics can easily express many of the properties that are needed for reasoning about concurrent systems. This is important because the reason some concurrency property holds is often quite subtle, and it is difficult to verify all possible cases manually.

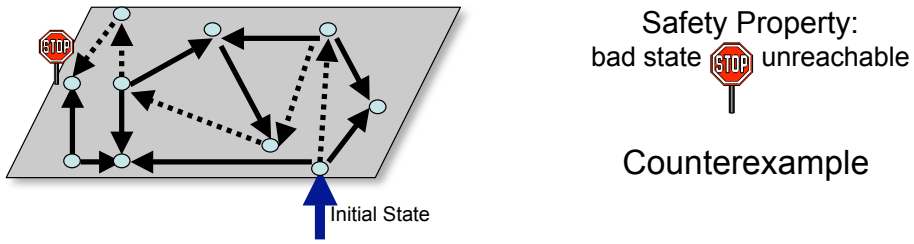


**Fig. 2.** Diagnostic Counterexample

### 1.3 Disadvantages of Model Checking

Over the last twenty-five years I have heard many objections to the use of Model Checking. I discuss some of these objections below:

– Proving a program helps you understand it. I do not believe that this is a valid objection. In my opinion it is somewhat like the saying that "Suffering makes us stronger". It is possible to understand a program just as well, if not better, by checking properties and examining the counterexamples when they are false.
– Temporal logic specifications are ugly. I think this depends on who is writing the specifications. I have seen very complicated and unreadable specifications in languages designed for formal specification based on Z (Zed) notation [ASM80]. A good rule of thumb is to keep the specifications as short as possible. Some model checkers have a macro facility that allows the user to encapsulate sub-expressions of formulas that would otherwise make it complicated. Temporal logics like PSL [EF06] have very expressive sets of operators that facilitate writing specifications.
– Writing specifications is hard. This is true. But it is also true of other verification techniques like automated theorem proving. Certainly, part of the solution is better education. Very few computer science and electrical engineering departments currently offer courses on formal verification. (Electrical engineers in the U.S. often spend more time learning about the Laplace transform than writing formal specifications for circuits!)
– State explosion is a major problem. This is absolutely true. The number of global system states of a concurrent system with many processes or complicated data structures can be enormous. All Model Checkers suffer from this problem. In fact, the state explosion problem has been the driving force behind much of the research in Model Checking and the development of new Model Checkers.

## 2 Verifications Tools Before 1981

Automated verification tools in use before 1981 were either based on theorem proving or exhaustive state exploration. I will focus on the state exploration techniques since they are more closely related to Model Checking.

### 2.1 Petri Net Tools

When I started research on this paper, I was certain that there had been earlier work on tools for verifying Petri Nets. I contacted two researchers, Tadao Murata and Kurt Jensen, who were active in the Petri Net community in the 1970's. To my surprise, I quickly discovered that there had been little serious work on verification tools for Petri Nets before 1981. I include brief quotes from Murata and Jensen below.

Murata:

> "I started working on Petri nets from mid-1970, and attended the First International Workshop on Petri Nets held in 1980 and thereafter. But

I do not recall any papers discussing formal verification using Petri Nets (PNs) BEFORE 1981. Also, I doubt there were any PN reachability tools before 1981. MetaSoft Company was selling earlier PN drawing tools and may have had a primitive one before 1981."

Jensen:

"Like Tad, I do not think there is any work on Petri net tools prior to 1981.The first Meta Software tool was made in the mid 80's and was merely a drawing tool for low level Petri nets. High-level Petri nets were invented in the late 70's. The first two publications appeared in TCS in 1979 and 1980. It is only after this that people really started the construction of tools. The first simulator for high-level nets and the first state space tools for these were made in the late 80's."

### 2.2 Bochmann and Protocol Verification

Around 1980, I became aware of the use of automatic verification techniques based on exhaustive state exploration by researchers in communication protocol verification. In particular, I read several very interesting papers by Gregor Bochmann. While researching my **25 MC** presentation, I contacted Bochmann and asked him to comment about his work on this topic. I enclose below a quote from his email message:

Bochmann:

"For a workshop organized by Andre Danthine, I prepared the paper **Finite State Description of Protocols** in which I presented a method for the verification of communication protocols using the systematic exploration of the global state space of the system (sometimes called reachability analysis). This paper was later published in Computer Networks (1978) and was much cited. At the same time, Colin West had developed some automated tools for doing essentially the same as what I was proposing, but I learned about his activities only later."

In the same message Bochmann commented about the importance of Model Checking.

"The need for exploring the reachable state space of the global system is the basic requirement in protocol verification. Here model checking has not provided anything new. However, temporal logic has brought a more elegant way to talk about liveness and eventuality; in the protocol verification community we were talking about reachable deadlock states (easy to characterize) or undesirable loops (difficult to characterize)."

I believe that Bochmann's comment is very perceptive, although I disagree with his statement that Model Checking has not contributed to the task of computing the reachable state space of a protocol. Indeed, much of the research in Model

Checking has focused on finding efficient techniques computing and representing the set of reachable states. Symbolic Model Checking [BCM+90], for example, was a major breakthrough because it enabled much larger state spaces to be searched than was possible using explicit state space traversal.

### 2.3 Holzmann and Protocol Verification

I was not aware of Gerard Holzmann's work on protocol verification until the late 1980's. In preparing for my MC 25 presentation, I contacted him to find out about his early work on automatic techniques for protocol verification.

Holzmann:

> "My first paper-method (never implemented) was from 1978-1979 – as part of my PhD thesis work in Delft. My first fully implemented system was indeed the *Pan* verifier (a first on-the-fly verification system), which found its first real bug in switching software (based on a model that I built in the predecessor language to Spin's *Promela*) at AT&T on November 21, 1980."

Spin did not use temporal logic for specifications until 1987 or 1988 and thus was not a true Model Checker in the sense that Emerson and I used the term until the late 1980's.

Holzmann continued:

> "Things changed quite a bit towards the late eighties, with machines getting faster and RAM memory larger. I implemented a small set of temporal properties (inspired by Pnueli's **Tools and Rules for the Practicing Verifier**) that expressed liveness in my verification system for SDL (the first such system built) in 1987/1988. That led to Spin in 1989 which generalized the method and allowed correctness properties to be expressed as unrestricted omega-regular properties (i.e., as never claims). The first full Spin version is from 1989. The converter from LTL to never claims was later designed by Doron, I think around 95, to make it easier for users to express LTL formulae directly."

Holtzmann argues that a Model Checker need not provide a logic for writing specifications.

> "When do we call an efficient checker that uses models a Model Checker though? I sometimes use the distinction between Model Checker and Logic Model Checker" – where to qualify for the latter term you need to support a logic."

I believe that Holzmann does have a valid point. Verification tools that compute some representation for the set of reachable states are often called Model Checkers as are sequential equivalence checkers in hardware verification. This is reasonable to me, although the term is not used in the spirit that Emerson and I originally intended.

## 3 Fixpoint Theory, Hoare Logic, and Concurrency

There is a close relationship between fixpoint theory and Model Checking algorithms for Branching-Time Logics. I read many papers on this topic as background research for my Ph.D. thesis. Perhaps the two most important results for my subsequent research on Model Checking were Tarski's Fixpoint Lemma [Tar55] and Kleene's First Recursion Theorem [Kle71]. Most Symbolic Model Checkers exploit Tarski's Lemma [Tar55] that every monotonic functional on a complete lattice has a fixpoint. A paper by David Park **Finiteness is Mu-Ineffable** [Par74] gives a first-order version of the Mu-calculus that I suggested as the logical basis for the first paper on Symbolic Model Checking that Burch, Dill, McMillan and I published in the 1990 LICS conference [BCM$^+$90,BCM$^+$92].

My first paper with Emerson [EC80] made the connection between Branching-Time Logics and the Mu-calculus. Kozen references the 1980 paper that Emerson and I wrote in his influential paper on the propositional Mu-calculus [Koz83].

Because of the close connection between the Mu-Calculus and Branching-time Temporal Logics, I believe it was inevitable that Model Checking algorithms were developed for Branching-time Logics before Linear-time Logics.

### 3.1 Thesis Research on Hoare Logic

My thesis dealt with the *Soundness* and *Completeness* of *Hoare Logic*. The two papers that influenced me most were:

- J. deBakker and L. Meertens, **On the Completeness of the Inductive Assertion Method**, [dBM75].
- S. Cook, **Soundness and Completeness of an Axiom System for Program Verification**, [Coo78].

Cook's paper introduced the notion of *Relative Completeness* of Hoare Logics.

I started on my thesis, entitled **Completeness and Incompleteness Theorems For Hoare Logics**, in July 1975 and finished it a year later in August 1976. Robert Constable was my advisor at Cornell. I waited until I had completed my thesis before publishing any papers on my research. I wrote three papers based on my thesis:

- E. Clarke, **Programming Language Constructs for which it is impossible to obtain Good Hoare-like Axiom Systems**, [Cla77b,Cla79c].
- E. Clarke, **Program Invariants as Fixedpoints**, [Cla77a,Cla79a].
- E. Clarke, **Proving Correctness of Coroutines Without History Variables**, [Cla78,Cla80].

In later research, I addressed the question of what programming language constructs could have *good* Hoare axiomatizations, i.e., sound and relatively complete axiomatizations.

- E. Clarke, S. German, J. Halpern, **Effective Axiomatizations of Hoare Logic**, [CGH83].

– E. Clarke, **Characterization Problem for Hoare Logics**, [Cla85].

The paper with German and Halpern gives a necessary and sufficient condition for the existence of a sound and relatively complete Hoare axiomatization. The 1985 paper gives a unified account of my research on Hoare logic and extends the results to *total correctness*.

### 3.2 Program Invariants as Fixed Points

In my thesis I showed that soundness and relative completeness results are really fixed point theorems. I gave a characterization of program invariants as fixed points of functionals obtained from the program text. For example, let $b * A$ denote **while b do A**. Let $wp[S](P)$ be the *weakest precondition for partial correctness* of the Predicate $P$ and the programming language statement $S$. Thus, $wp[S](P)$ satisfies two properties:

1. The Hoare triple $\{\, wp[S](P) \,\}\ S\ \{\, P \,\}$ is true in the logical structure under consideration, and
2. If the triple $\{\, P \,\}\ S\ \{\, Q \,\}$ is true, then $P \rightarrow wp[S](Q)$ is true.

It is not difficult to prove

$$wp[b * A](Q) = (\neg b \wedge Q) \vee (b \wedge wp[A](wp[b * A](Q)))$$

Thus, $wp[b * A](Q)$ is a fixpoint of the functional

$$\tau(U) = (\neg b \wedge Q) \vee (b \wedge wp[A](U)).$$

In fact, $wp[b * A](Q)$ is the greatest fixpoint of the functional $\tau$. The fixpoint characterizations are more complicated for programming language constructs that are not tail recursive.[1]

I showed that Relative Completeness is logically equivalent to the *existence* of a fixed point for an appropriate functional, and that Relative Soundness follows from the *maximality* of the fixed point.

For finite interpretations, the results give a decision procedure for partial correctness, i.e., a primitive Model Checker for partial correctness! When I originally proved these results, this idea occurred to me, but I thought it would not be practical and did not pursue the idea further at the time.

[1] I was unaware of the work by Basu and Yeh [BY75] until I saw it cited in Emerson's paper in this volume. The paper shows that the weakest precondition for total correctness is the least fixed point of a functional obtained from the body of a while loop. The theory in my thesis and the papers mentioned above applies to partial correctness as well as total correctness and handles general loops (regular recursions) and non-regular recursions as well. I also relate my fixpoint theory to relative soundness and completeness proofs of Cook and others.

### 3.3 Data-flow Analysis

In 1978, I moved to Harvard. At Harvard, I taught the undergraduate course on Compilers. In preparing for this course, I read a number of papers on *data-flow analysis* including:

- G. Killdall, **A Unified Approach to Global Program Optimization**, [Kil73].
- J. B. Kam and J. D. Ullmann, **Monotone Data-flow Analysis Frameworks**, [KU77].
- Richard N. Taylor and Leon J. Osterweil, **Anomaly Detection in Concurrent Software by Static Data Flow Analysis**, [TO80].
- P. Cousot and R. Cousot, **Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints**, [CC77].

The paper by Fosdick and Osterweil was definitely ahead of its time. Although it was written thirty years ago, the title sounds surprisingly modern. In fact, several papers with similar sounding titles have been published in recent CAV and TACAS conferences.

Data-flow analysis can be considered to be an instance of Model Checking as the 1998 paper by David Schmidt demonstrates:

- D. Schmidt, **Data-flow Analysis is Model Checking of Abstract Interpretations**, [Sch98].

### 3.4 My Early Research On Concurrency

In 1977 I read the classic paper by Owicki and Gries [OG76] on methods for reasoning about concurrent systems using *conditional critical regions* for synchronization. My research focussed on fixpoint equations, abstract interpretation, and widening for concurrent programs. This research led to three papers:

- E. M. Clarke, **Synthesis of Resource Invariants for Concurrent Programs**, [Cla79b].
- E. Clarke and L. Liu, **Approximate Algorithms for Optimization of Busy Waiting in Parallel Programs**, [CL79].
- L. Liu and E. Clarke, **Optimization of Busy Waiting in Conditional Critical Regions**, [LC80].

## 4 Temporal Logic

Temporal logics describe the ordering of events in time without introducing time explicitly. They were developed by philosophers and linguists for investigating how time is used in natural language arguments. Most temporal logics have an operator like $Gf$ that is true in the present if $f$ is always true in the future. To assert that two events $e_1$ and $e_2$ never occur at the same time, one would write $G(\neg e_1 \vee \neg e_2)$. Temporal logics are often classified according to whether time is assumed to have a linear or a branching structure. The meaning of a temporal logic formula is determined with respect to a labeled state-transition graph or *Kripke structure*.

### 4.1 Temporal Logic and Program Verification

Burstall [Bur74], Kröger [Krö77], and Pnueli [Pnu77], all proposed using temporal logic for reasoning about computer programs. Pnueli was the first to use temporal logic for reasoning about concurrency. He proved program properties from a set of axioms that described the behavior of the individual statements. The method was extended to sequential circuits by Bochmann [Boc82] and Malachi and Owicki [MO81]. Since proofs were constructed by hand, the technique was often difficult to use in practice.

### 4.2 Pnueli's 1977 Paper and Model Checking

I reread Pnueli's 1977 paper [Pnu77] in preparing for my 25MC lecture. The section entitled Finite State Systems is extremely interesting, although I do not remember reading it before Emerson and I wrote our 1981 paper [CE81]. After rereading this section, an obvious question is whether Pnueli should be credited with inventing Model Checking in 1977. Theorems 4 and 5 in his paper are particularly noteworthy.

> **Theorem 4**: The validity of an arbitrary eventuality $G(A \rightarrow FB)$ is decidable for any finite state system.

The proof of this theorem uses strongly connected components and is very similar to the technique used for EG(P) in CES 83/86 [CES83,CES86]. Theorem 5 is quite general.

> **Theorem 5**: The validity of an arbitrary tense formula on a finite state system is decidable and the extended system Kb is adequate for proving all valid (propositional) tense formulas.

The proof of Theorem 5 is briefly discussed in an appendix to Pnueli's paper.

> Theorem 5 may be proved by reduction of the problem of validity of a propositional tense formula on a finite state system to that of the validity of a formula in the Monadic Second Order Theory of Successor.
>
> We show that for each propositional tense formula formula $W$, we can construct an $\omega$-regular language $L(W)$ which describes all those $S^\omega$ sequences on which $W$ is true.
>
> Our decision problem reduces to the question is $L(A_\Sigma) \subseteq L(W)$, i.e. do all proper execution sequences of $A_\Sigma$ satisfy $W$.

The reference that Pnueli [Pnu77] gives for checking containment of $\omega$-regular languages does not indicate how an efficient algorithm could be constructed for this purpose. Clearly, if Pnueli did discover Model Checking in 1977, he also discovered *Automata Theoretic Model Checking* at the same time.

### 4.3 Branching-Time Logics

Emerson and I [EC80] proposed a very general branching-time temporal logic based on *Computation Trees* (Figure 3) and made the connection with the mu-calculus. Ben-Ari, Manna, and Pnueli (81 / 83) [BAMP83] gave an elegant syntax for a branching time logic called UB. Here is how *inevitably p* would be expressed in both logics. In EC 80, we wrote $\forall$ path $\exists$ node $p$. The notation in BMP 81 was much more concise. They simply wrote AF $p$. In [CE81] we adopted the UB notation and introduced two versions of the until operator ($AU$ and $EU$).

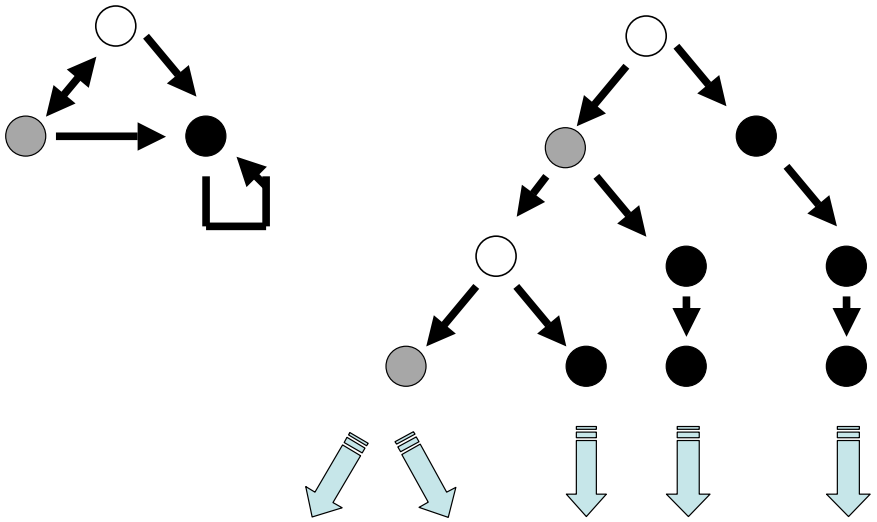

**Fig. 3.** Kripke Structure and Computation Tree

### 4.4 Expressive Power of Temporal Logic

Lamport was the first to investigate the expressive power of various temporal logics for verification. His 1980 POPL paper [Lam80] discusses two logics: a simple linear-time logic and a simple branching-time logic. He showed that each logic could express certain properties that could not be expressed in the other. Branching-time logic cannot express certain natural fairness properties that can be easily expressed in the linear-time logic. Linear-time logic cannot express the possibility of an event occurring sometime in the future along some computation path. Technical difficulties made Lamport's result somewhat like comparing "apples and oranges".

Emerson and Halpern [EH86] provided a uniform framework for investigating this question. They formulated the problem in terms of a single logic called CTL*, which combines both linear-time and branching-time operators. A state formula may be obtained from a path formula by prefixing it with a path quantifier, either be an $A$ (for every path) or an $E$ (there exists a path). Linear-time logic (LTL) is identified with the set of CTL* state formulas $Af$ where $f$ is a path formula not containing any state sub-formulas. The branching-time part (CTL) consists of all state formulas in which every linear-time operator is immediately preceded by a path quantifier. Since both LTL and CTL consist entirely of state formulas, they were able to avoid the uniform framework problem in Lamport's paper.

They showed that there exists a formula of LTL that cannot be expressed in CTL and vice versa. In general, the proofs of their inexpressibility results are quite long and tedious. For example, the proof that the linear-time formula $A(FGp)$ is not expressible in CTL uses a complicated inductive argument that requires 3.5 journal pages to present. Furthermore, the technique that they use does not easily generalize to other examples.

In [CD88], Anca Dragahicescu (now Browne) proved the following theorem:

> **Theorem**: Let $M = (S, R, L, F)$ be a Kripke Structure with Müller Fairness Constraints, and let $M' = (S, R, L, F')$ where the set of constraints $F'$ extends $F$. Then for all CTL formulas $f$ and all states $s \in S$, $M, s \models f$ if and only if $M', s \models f$.

We used the theorem to give a short proof that no CTL formula can express $A(FG\,p)$ for the special case of Kripke structures with Müller fairness constraints.

## 5 Temporal Logic Model Checking

The basic papers on the use of Temporal Logic Model Checking were written in the early 1980's. I describe what was done and comment on similarities and differences between various approaches.

### 5.1 Clarke and Emerson 1981

My work with Emerson came first in the spring of 1981 [CE81]. It was presented in a predecessor conference of LICS organized by Dexter Kozen.

- Edmund M. Clarke and E. Allen Emerson,
- Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic.
- Presented at the Logics of Programs Workshop at Yorktown Heights, New York in May 1981.
- The proceedings were published in LNCS 131.
- Also in Emerson's 1981 Ph.D. Thesis.

The temporal logic model checking algorithms that Emerson and I developed allowed this type of reasoning to be automated. Checking that a single structure satisfies a formula is much easier than proving the validity of a formula for all structures. Our algorithm for CTL was polynomial in the product of $|M|$ and $|f|$. We also showed how fairness could be handled without changing the complexity of the algorithm.

Emerson and I had a Harvard undergraduate (Marshall Brin) implement the fixpoint algorithm for Model Checking. Unfortunately, the implementation was incorrect. There was a problem with fairness constraints. To our embarrassment, we discovered this when we demonstrated the Model Checker to Bochmann when he gave a lecture at Harvard.

### 5.2 My Eureka Moment

In the fall of 1980 and the spring of 1981, Emerson was writing his Ph.D. thesis on the synthesis of finite state concurrent programs from CTL specifications. The idea was to use a decision procedure for satisfiability of CTL formulas to extract a finite model from a specification in CTL. The concurrent program could then be extracted from the finite model. There were two disadvantages to this approach: the exponential complexity of the decision procedure (in practice as well as theory) and the need to completely specify the concurrent program in temporal logic.

In January of 1981, I attended POPL where the paper by Ben-Ari, Manna, and Pnueli [BAMP83] on "The Temporal Logic of Branching Time" was originally presented. I had trouble understanding non-trivial formulas of the logic. I spent several hours drawing Kripke Structures and checking to see if various formulas were true or not. If the structures had many states and the formulas were complicated, this turned out to be more complicated than I expected, and my first guess was often wrong. I tried to find an algorithm to automate this process. For some operators like AF p, the algorithm was obvious–just perform a depth-first search starting from the initial state of the structure and see if there was a path ending in a cycle along which $\neg p$ always held. I suspected that there was a linear algorithm for the problem, but getting it correct for all of CTL was tricky.

After trying several examples, it occurred to me that often complex communication and synchronization protocols were specified by state machines and that an efficient algorithm for the checking formulas on models could be used to see if the state machines satisfied their specifications. This was my "Eureka moment"! I realized that the important problem for verification was not the synthesis problem but the problem of checking formulas on finite models. I began to work on a depth-first search algorithm for the Model Checking problem. When I told Emerson about my conclusion, he saw how fixpoint techniques could be used to obtain an algorithm for the complete logic that was quadratic in the size of the model. Of course, the quadratic complexity of the algorithm meant that it did not scale to large models. I doubt if it would have been able to handle a model with a 1000 states.

In the fall of 1982 after my move to Carnegie Mellon, I developed a strictly graph theoretic algorithm for CTL Model Checking with Fairness Constraints. My algorithm had linear complexity in the size of the model. I implemented the algorithm myself in the EMC Model Checker. I wrote the program in a language called "Franz Lisp" and still have the original code! The new implementation is described in my 1983 POPL paper with Emerson and Sistla [CES83,CES86].

### 5.3 Quielle and Sifakis 1982

The work of Quielle and Sifakis was presented at a conference in the Spring of 1982 [QS82], although a technical report version appeared in June of 1981. I learned about their research when I was working on CES 83 / 86 probably late in the fall of 1982. Their work was certainly independent of ours. I regard this as a case of essentially simultaneous discovery of an idea whose time was ripe.

- J.P. Queille and J. Sifakis
- Specification and Verification of Concurrent Systems in Ceasar
- Technical Report 254 June 1981
- International Symposium on Programming, Turin, April, 1982
- Springer Lecture Notes in Computer Science 137, published in 1982

There are a number of similarities between the work that Emerson and I did and the work of Quielle and Sifakis:

- Both used a branching-time temporal logic related to [BAMP83]. (**POT** is like **EF** and **INEV** is like our **AF**.)
- Formula evaluation in [QS82] is by computing fixpoints as in [CE81]. In [CES83,CES86] more efficient graph algorithms are used.
- The programming language CSP [Hoa85] is used for describing models in both [QS82] and [CES83,CES86]. The Alternating Bit Protocol [BSW69] is also used for illustration in both [QS82] and [CES83,CES86].
- There is a clear distinction between the model and the formula to be checked in both (the term "Model Checking" originates with [CE81], however).

There are also a number of important differences:

- The logic used in [QS82] does not have an *until* operator **U** (trivial).
- Quielle and Sifakis do not analyze the complexity of their algorithm.
- Finally, they did not implement *fairness constraints*.

Their paper references Clarke [Cla77a,Cla79a] and Cousot [CC77] for computing fixed points of monotonic operators on a lattice [2].

[2] I sent a draft of this paper to Sifakis. He replied that they had another paper in FOCS 1982 and Acta Inf. 1983 [QS83] that included the *until* operator and could express a particular class of fairness properties. However, this paper references [CE81], and after 25 years, Sifakis was unable to explain how it differed from our first paper.

### 5.4 The EMC Model Checker

My paper with Emerson and Sistla [CES83,CES86] gave an improved algorithm that was linear in the product of the $|M|$ and $|f|$. The algorithm was implemented in the EMC Model Checker and used to check a number of network protocols and sequential circuits (EMC stands for *Extended Model Checker*. At the risk of being obvious, note the similarity to my initials). It could check state transition graphs with between $10^4$ and $10^5$ states at a rate of about 100 states per second for typical formulas. In spite of these limitations, EMC was used successfully to find previously unknown errors in several published circuit designs.

The EMC Model Checker was the first Model Checker to implement *Fairness Constraints*. Fairness Constraints are formulas that must hold infinitely often on each fair path. This feature made it possible to check some important properties that could not be expressed in CTL. An example of such a property is $A(GF\,enabled \rightarrow GF\,executed)$, which expresses that property that a process that is enabled for execution infinitely often must actually be executed infinitely often. Because of this feature, the EMC algorithm was able to solve the *Emptiness Problem for Non-deterministic Büchi Automata* in time linear in the size of the automaton.

**Hardware Verification** My student, Bud Mishra, was the first to use Model Checking for Hardware Verification [MC85]. He found a bug in the Sietz FIFO Queue (Figure 4) from Mead and Conway's book, Introduction to VLSI Systems [MC79]. David Dill and Mike Browne also started working on hardware verification. The four of us wrote several papers on applying Model Checking to hardware verification [MC85,BCD85,BCD86,BCDM86,DC86]

**Witnesses and Counterexamples** EMC did not give counterexamples for universal CTL properties that were false or witnesses for existential properties that were true. I asked my student, Michael C. Browne, to add this feature to the MCB model Checker in 1984 (MCB stands for *Model Checker B*. However, note the similarity to Browne's initials). It has been an important feature of Model Checkers ever since (Figure 5).

### 5.5 LTL and CTL*

**Complexity of LTL** Sistla and I [SC86] analyzed the model checking problem for LTL and showed that the problem was PSPACE-complete. Pnueli and Lichtenstein [LP85] gave an algorithm that is exponential in the length of the formula, but linear in the size of the Model. Based on this observation, they argued that LTL model checking is feasible for short formulas.

**CTL* Model Checking** CTL* is a very expressive logic that combines both branching-time and linear-time operators. Model checking for this logic was first considered in[CES83,CES86] where it was shown to be PSPACE-complete.

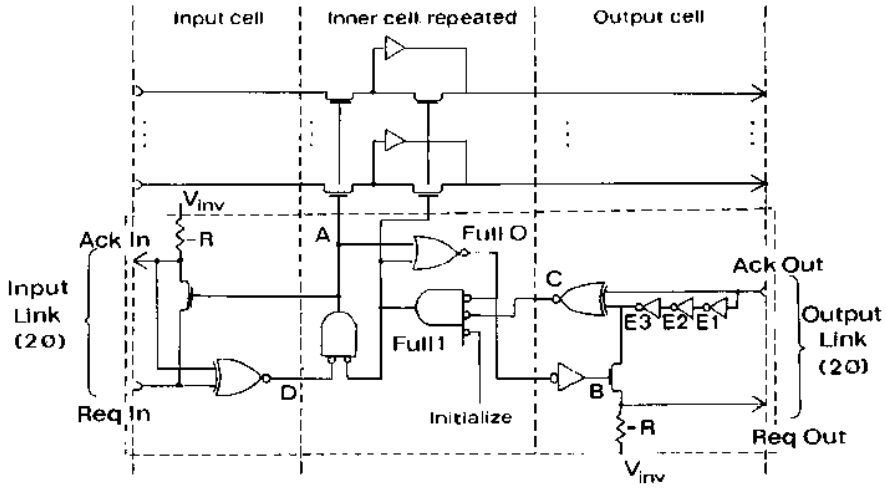


**Fig. 4.** Self-Timed FIFO Queue from Mead and Conway

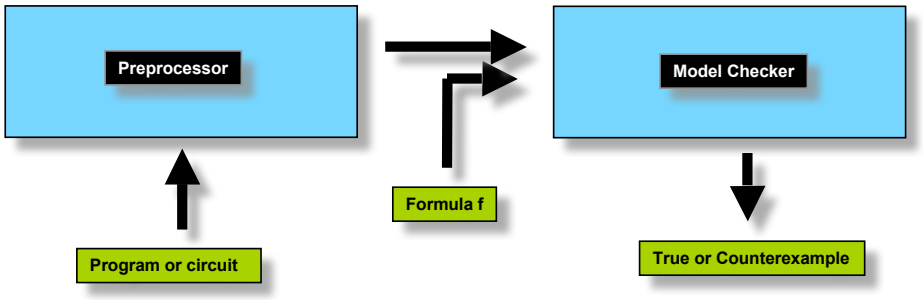


**Fig. 5.** A Model Checker that supports Counterexamples

Emerson and Lei [EL85] showed that CTL* and LTL Model Checking have the same complexity in $|M|$ and $|f|$. Thus, for purposes of Model Checking, there is no practical complexity advantage to restricting oneself to a linear temporal logic.

### 5.6 Automata Theoretic Techniques and Process Algebra

**Using Automata for both Models and Specifications** Alternative techniques for verifying concurrent systems have been proposed by a number of other researchers. Some approaches use automata for specifications as well as for implementations. The implementation is checked to see whether its behavior conforms to that of the specification. Thus, an implementation at one level can be used as a specification for the next level. The use of language containment is implicit in the work of Kurshan, which ultimately resulted in the development of the COSPAN verifier [AKS83,HK87,Dil89].

**Automata Theoretic Model Checking with LTL** Vardi and Wolper [VW86] first proposed the use of $\omega$-automata (automata over infinite words) for automated verification. They showed how the linear temporal logic Model Checking could be formulated in terms of language containment. Many explicit state LTL Model Checkers (e.g., Spin) use a variant of this construction. It can also be used with Symbolic and Bounded Model Checkers as well.

**Links to Process Algebra** If two finite Kripke Structures can be distinguished by some CTL* formula, then they can be distinguished by a CTL formula. In [BCG88] we showed that for any finite Kripke structure $M$, it is possible to construct a CTL formula $F_M$ that uniquely characterizes $M$. We use a notion of equivalence between Kripke Structures, similar to the notion of bisimulation studied by Milner [Mil71] and Park [Par81]. The first construction of $F_M$ uses of the next-time operator $X$. We also considered the case in which the next-time operator is disallowed. The proof, in this case, required another notion of equivalence, *equivalence with respect to stuttering*. We gave a polynomial algorithm for determining if two structures are stuttering equivalent.

## 6 Dealing with Very Complex Systems

Significant progress was made on the State Explosion Problem around 1990. Both Symbolic Model Checking and the Partial Order Reduction were developed about this time.

### 6.1 Symbolic Model Checking

In the original implementation of the Model Checking algorithm, transition relations were represented explicitly by adjacency lists. For concurrent systems with

small numbers of processes, the number of states was usually fairly small, and the approach was often quite practical. In systems with many concurrent parts the number of states in the global state transition graph was too large to handle. In the fall of 1987, McMillan, then a graduate student of mine at Carnegie Mellon, realized that by using a symbolic representation for the state transition graphs, much larger systems could be verified. The new symbolic representation was based on *ordered binary decision diagrams* (OBDDs) [BCM+90,McM93]. OBDDs provide a canonical form for boolean formulas that is often substantially more compact than conjunctive or disjunctive normal form, and very efficient algorithms have been developed for manipulating them. Because the symbolic representation captures some of the regularity in the state space determined by circuits and protocols, it is possible to verify systems with an extremely large number of states—many orders of magnitude larger than could be handled by the explicit-state algorithms. By using the original CTL Model Checking algorithm of Clarke and Emerson with the new representation for state transition graphs, it became possible to verify some examples that had more than $10^{20}$ states. Since then, various refinements of the OBDD-based techniques by other researchers have pushed the state count up to more than $10^{120}$.

**The SMV Model Checker** The Model Checking system that McMillan developed as part of his Ph.D. thesis is called SMV [McM93]. It is based on a language for describing hierarchical finite-state concurrent systems. Programs in the language can be annotated by specifications expressed in temporal logic. The Model Checker extracts a transition system represented as an OBDD from a program in the SMV language and uses an OBDD-based search algorithm to determine whether the system satisfies its specification. If the transition system does not satisfy some specification, the verifier will produce an execution trace that shows why the specification is false. The SMV system has been widely distributed, and a large number of examples have now been verified with it. These examples provide convincing evidence that SMV can be used to debug real industrial designs. Now there are two widely used versions of SMV: Cadence SMV released by Cadence Berkeley Labs and an open source version, called *NuSMV* [CCGR00], released by IRST in Trento, Italy.

Verification of the cache coherence protocol in the IEEE Futurebus+ standard illustrates the power of the SMV Model Checker. Development of the protocol began in 1988, but all previous attempts to validate it were based on informal techniques. In the summer of 1992 my group at Carnegie Mellon constructed a precise model of the protocol. Using SMV we were able to find several previously undetected errors and potential errors in the design of the protocol. This was the first time that an automatic verification tool had been used to find errors in an IEEE standard [CGH+93,Lon93].

**Other Work on Symbolic Model Checking** Several other researchers independently discovered that OBDDs can be used to represent state-transition systems. Coudert, et al. [CBM89] gave an algorithm for sequential equivalence

checking that used OBDDs for the transition functions. Bose and Fisher [BF89], Pixley [Pix90], and Coudert et al. [CBM90] also experimented with symbolic Model Checking algorithms.

### 6.2 Partial Order Reduction

Verifying software causes some problems for Model Checking. Software tends to be less structured than hardware. In addition, concurrent software is usually *asynchronous*, i.e., most of the activities taken by different processes are performed independently, without a global synchronizing clock. For these reasons, the state explosion phenomenon is a particularly serious problem for software. Consequently, Model Checking has been used less frequently for software verification than for hardware verification. The most successful techniques for dealing with asynchronous systems are based on the *partial order reduction*. These techniques exploit the independence of concurrently executed events. Two events are *independent* of each other when executing them in either order results in the same global state.

Model Checking algorithms that incorporate the partial order reduction are described in several different papers. The *stubborn sets* of Valmari [Val90], the *persistent sets* of Godefroid [God90] and the *ample sets* of Peled [Pel94] differ on the actual details, but contain many similar ideas. Other methods that exploit similar observations about the relation between the partial and total order models of execution are McMillan's *unfolding technique* [McM93] and Godefroid's *sleep sets* [God90].

### 6.3 Special Purpose Techniques

Special techniques are needed when symbolic methods and the partial order reduction don't work. Four basic techniques are

- Compositional Reasoning,
- Abstraction,
- Symmetry Reduction,
- Induction and Parameterized Verification.

**Compositional Reasoning** This technique exploits the modular structure of complex circuits and protocols. Many finite state systems are composed of multiple processes running in parallel. The specifications for such systems can often be decomposed into properties that describe the behavior of small parts of the system. An obvious strategy is to check each of the local properties using only the part of the system that it describes. If the system satisfies each local property, and if the conjunction of the local properties implies the overall specification, then the complete system must satisfy this specification as well.

The naive form of compositional reasoning may not be feasible because of mutual dependencies between the components. When verifying a property of

one component assumptions are needed about the behavior of the other components. The assumptions must later be discharged when the correctness of the other components is established. This strategy is called *assume-guarantee reasoning* [MC81,Jon83,Pnu84,GL94].

The main problem in employing assume-guarantee style reasoning in verification relates to effectively computing environment assumptions for each component. Initial attempts to perform such reasoning focused on hardware systems [McM97,AH99] and the assumptions were provided manually. Recently, a method for automatically generating these assumptions has been proposed [CGP03]. Here, the task of computing an assumption is posed as a machine learning problem, where a learning algorithm for regular languages $L^*$ [Ang87,RS93] is used to generate a finite-state assumption in an iterative fashion by making queries to a *teacher* entity. A model checker plays the role of the teacher and assists the learner by answering queries and providing counterexamples. Extensions of this approach have been used to solve the *Component Substitutability Problem* [CCSS05,CCST05]. A symbolic extension of this approach using BDDs has also been proposed [AMN05].

**Abstraction** Abstraction is essential for reasoning about reactive systems that involve data. It is based on the observation that specifications of systems usually involve simple relationships among data values. For example, verifying a program may depend on simple arithmetical relationships (predicate abstraction). In such situations abstraction can be used to reduce the complexity of Model Checking. The abstractions is usually specified by a mapping between data values in the system and a small set of abstract data values. By extending the mapping to states and transitions, it is possible to produce a much smaller, abstract version [CGL92,BBLS92,CGL94].

**Symmetry Reduction** Symmetry [ID93,CFJ93,ES93] can be used to reduce the state explosion problem. Finite state concurrent systems often contain replicated components, e.g., a network of identical processes communicating in some fashion. This information can be used to obtain reduced models. Having physical symmetry in a system often implies existence of a non-trivial permutation group that preserves the transition graph. The permutation group can be used to define an equivalence relation on the state space. The resulting reduced model can be used to simplify verification of Temporal logic properties.

**Parameterized Systems** Induction involves reasoning automatically about entire families of finite state systems. Typically, circuit and protocol designs are parameterized, that is, they define an infinite family of systems. For example, a bus protocol may be designed for an arbitrary number of processors. Ideally, one would like to be able to check that every system in a given family satisfies some temporal logic property. In general, the problem is undecidable [AK86]. Often it is possible to provide an invariant process that represents the behavior of an

arbitrary member of the family. Using the invariant, one can check property for all members of the family at once [CGB86,KM89,WL89].

## 7 Big Events Since 1990 and Future Challenges

The pace of research in Model Checking has accelerated since 1990. Below I list several of the most important breakthroughs during this period. I only cite the initial paper (or papers) that led to the breakthrough, although each of the seminal papers led to many papers often containing significant extensions of the original work.

- Timed and Hybrid Automata [ACD90,HKPV95]
- Model Checking for Security Protocols [Ros94,MCJ97]
- Bounded Model Checking [BCCY99,BCC+03]
- Localization Reduction and CEGAR [Kur94,CGJ+00]
- Compositional Model Checking and Learning [MC81,Jon83,Pnu84,GL94]
- Predicate Abstraction [GS97,BMMR01]
- Infinite State Systems (e.g., pushdown systems) [BEM97]

I conclude with a list of challenges for the future. I believe that all of the problems in the list are important and that all require major breakthroughs in order to become sufficiently practical for widespread use in industry.

- Software Model Checking, Model Checking and Static Analysis
- Model Checking and Theorem Proving
- Exploiting the Power of SAT, Satisfiability Modulo Theories (SMT)
- Probabilistic Model Checking
- Efficient Model Checking for Timed and Hybrid Automata
- Interpreting Counterexamples
- Coverage (incomplete Model Checking, have I checked enough properties?)
- Scaling up even more!!

I expect the next twenty-five years will hold many surprises and be at least as exciting as the past twenty-five. I look forward with great enthusiasm to participating in at least some of this research.

**Acknowledgements.** The author wishes to thank Nishant Sinha for his help in preparing this document. Martha Clarke, Jonathan Clarke, and Katie Clarke read early versions of this document and gave useful comments.

## References

[ACD90] R. Alur, C. Courcourbetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the 5th Symp. on Logic in Computer Science*, pages 414–425, 1990.

[AH99] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design: An International Journal*, 15(1):7–48, July 1999.

[AK86] K. Apt and D. Kozen. Limits for automatic verification of finite-state systems. *IPL*, 15:307–309, 1986.

[AKS83] S. Aggarwal, R. P. Kurshan, and K. Sabnani. A calculus for protocol specification and validation. In H. Rudin and C. H. West, editors, *Protocol Specification , Testing and Verification*, pages 19–34. North Holland, 1983.

[AMN05] R. Alur, P. Madhusudan, and W. Nam. Symbolic compositional verification by learning assumptions. In *CAV*, 2005.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. In *Information and Computation*, volume 75(2), pages 87–106, November 1987.

[ASM80] Jean-Raymond Abrial, Stephen A. Schuman, and Bertrand Meyer. Specification language. In R. M. McKeag and A. M. Macnaughten, editors, *On the Construction of Programs*, pages 343–410. Cambridge University Press, 1980.

[BAMP83] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[BBLS92] S. Bensalem, A. Bouajjani, C. Loiseaux, and J. Sifakis. Property preserving simulations. In G. V. Bochmann and D. K. Probst, editors, *Proc. 4th Workshop on Comput.-Aided Verification*, pages 260–273, July 1992.

[BCC+03] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Y. Zhu. *Bounded Model Checking*, volume 58 of *Advances in computers*. Academic Press, 2003.

[BCCY99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Yhu. Symbolic model checking without BDDs. volume 1579 of *LNCS*, pages 193–207, March 1999.

[BCD85] M. C. Browne, E. M. Clarke, and D. Dill. Checking the correctness of sequential circuits. In *Proceedings of the 1985 International Conference on Computer Design*, pages 545–548, Port Chester, New York, October 1985. IEEE.

[BCD86] M. C. Browne, E. M. Clarke, and D. L. Dill. Automatic circuit verification using temporal logic: Two new examples. In *Formal Aspects of VLSI Design*. Elsevier Science Publishers (North Holland), 1986.

[BCDM86] M. C. Browne, E. M. Clarke, D. L. Dill, and B. Mishra. Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, C-35(12):1035–1044, 1986.

[BCG88] M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1–2):115–131, July 1988.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: $10^{20}$ states and beyond. In *Proc. 5th Ann. Symp. on Logic in Comput. Sci.* IEEE Comp. Soc. Press, June 1990.

[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, June 1992.

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *International Conference on Concurrency Theory*, pages 135–150, 1997.

[BF89] S. Bose and A. Fisher. Verifying pipelined hardware using symbolic logic simulation. In *IEEE International Conference on Computer Design*, October 1989.

[BMMR01] Thomas Ball, Rupak Majumdar, Todd D. Millstein, and Sriram K. Rajamani. Automatic predicate abstraction of C programs. volume 36(5), pages 203–213, June 2001.

[Boc82] G. V. Bochmann. Hardware specification with temporal logic: An example. *IEEE Transactions on Computers*, C-31(3), March 1982.

[BSW69] Keith A. Bartlett, Roger A. Scantlebury, and Peter T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Commun. ACM*, 12(5):260–261, 1969.

[Bur74] R. M. Burstall. Program proving as hand simulation with a little induction. In *IFIP congress 74*, pages 308–312. North Holland, 1974.

[BY75] Sanat K. Basu and Raymond T. Yeh. Strong verification of programs. *IEEE Trans. Software Eng.*, 1(3):339–346, 1975.

[CBM89] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In Sifakis [Sif89], pages 365–373.

[CBM90] O. Coudert, C. Berthet, and J. C. Madre. Verifying temporal properties of sequential machines without building their state diagrams. In Kurshan and Clarke [KC90], pages 23–32.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th Ann. ACM Symp. on Principles of Prog. Lang.*, pages 238–252, January 1977.

[CCGR00] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. Nusmv: A new symbolic model checker. *STTT*, 2(4):410–425, 2000.

[CCSS05] Sagar Chaki, Edmund Clarke, Natasha Sharygina, and Nishant Sinha. Dynamic component substitutability analysis. In *Proc. of Conf. on Formal Methods*, 2005.

[CCST05] Sagar Chaki, Edmund Clarke, Nishant Sinha, and Prasanna Thati. Automated assume-guarantee reasoning for simulation conformance. In *Proc. of Computer-Aided Verific*, 2005.

[CD88] E. M. Clarke and I. A. Draghicescu. Expressibility results for linear time and branching time logics. In *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, volume 354, pages 428–437. Springer-Verlag: Lecture Notes in Computer Science, 1988.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981*, volume 131 of *LNCS*. Springer-Verlag, 1981.

[CES83] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. In *Proc. 10th Ann. ACM Symp. on Principles of Prog. Lang.*, January 1983.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CFJ93] E. M. Clarke, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. In Courcoubetis [Cou93], pages 450–462.

[CGB86] E. M. Clarke, O. Grumberg, and M. C. Browne. Reasoning about networks with many identical finite-state processes. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing.*, pages 240–248. ACM, August 1986.

[CGH83] Edmund M. Clarke, Steven M. German, and Joseph Y. Halpern. Effective axiomatizations of hoare logics. *J. ACM*, 30(3):612–636, 1983.

[CGH+93] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the Futurebus+ cache coherence protocol. In Claesen [Cla93].

[CGJ+00] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169, 2000.

[CGL92] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. In *POPL*, pages 342–354, 1992.

[CGL94] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, September 1994.

[CGP03] J. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. volume 2619 of *LNCS*, pages 331–346, April 2003.

[CL79] Edmund M. Clarke and L. Liu. Approximate algorithms for optimization of busy waiting in parallel programs (preliminary report). In *20th Annual Symposium on Foundations of Computer Science*, pages 255–266. IEEE Computer Society, 1979.

[Cla77a] Edmund M. Clarke. Program invariants as fixed points (preliminary reports). In *18th Annual Symposium on Foundations of Computer Science*, pages 18–29. IEEE Computer Society, November 1977.

[Cla77b] Edmund M. Clarke. Programming language constructs for which it is impossible to obtain "good" hoare-like axiom systems. In *Fourth ACM Symposium on Principles of Programming Languages*, pages 10–20, New York, NY, USA, January 1977. ACM Press.

[Cla78] Edmund M. Clarke. Proving the correctness of coroutines without history variables. In *ACM-SE 16: Proceedings of the 16th annual Southeast regional conference*, pages 160–167, New York, NY, USA, 1978. ACM Press.

[Cla79a] Edmund Clarke. Program invariants as fixed points. *COMPUTING*, 21(4):273–294, 1979.

[Cla79b] Edmund M. Clarke. Synthesis of resource invariants for concurrent programs. In *POPL*, pages 211–221, 1979.

[Cla79c] Edmund Melson Clarke. Programming language constructs for which it is impossible to obtain good hoare axiom systems. *J. ACM*, 26(1):129–147, 1979.

[Cla80] Edmund Clarke. Proving correctness of coroutines without history variables. *Acta Inf.*, 13:169–188, 1980.

[Cla85] E. M. Clarke. The characterization problem for hoare logics. In *Proc. of a discussion meeting of the Royal Society of London on Mathematical logic and programming languages*, pages 89–106, Upper Saddle River, NJ, USA, 1985. Prentice-Hall, Inc.

[Cla93] L. Claesen, editor. *Proc. 11th Int. Symp. on Comput. Hardware Description Lang. and their Applications*. North-Holland, April 1993.

[Coo78] Stephen A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, 1978.

[Cou93] C. Courcoubetis, editor. *Proc. 5th Workshop on Comput.-Aided Verification*, June/July 1993.

[dBM75] Jaco W. de Bakker and Lambert Meertens. On the completeness of the inductive assertion method. *Journal of Computer and System Sciences*, 11:323–357, 1975.

[DC86] D. L. Dill and E. M. Clarke. Automatic verification of asynchronous circuits using temporal logic. *IEE Proceedings*, Part E 133(5), 1986.

[Dil89] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1989.

[EC80] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Lecture Notes in Computer Science 85*, pages 169–181. Automata, Languages and Programming, July 1980.

[EF06] Cindy Eisner and Dana Fisman. *A Practical Introduction to PSL (Series on Integrated Circuits and Systems).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited: On branching time versus linear time. *Journal of the ACM*, 33:151–178, 1986.

[EL85] E. A. Emerson and C-L. Lei. Modalities for model checking: Branching time strikes back. *Twelfth Symposium on Principles of Programming Languages, New Orleans, La.*, pages 84–96, January 1985.

[ES93] E. A. Emerson and A. P. Sistla. Symmetry and model checking. In Courcoubetis [Cou93], pages 463–478.

[GL94] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16:843–872, May 1994.

[God90] P. Godefroid. Using partial orders to improve automatic verification methods. In Kurshan and Clarke [KC90].

[GS97] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. volume 1254 of *LNCS*, pages 72–83, June 1997.

[HK87] Z. Har'El and R. P. Kurshan. The COSPAN user's guide. Technical Report 11211-871009-21TM, AT&T Bell Labs, 1987.

[HKPV95] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual Symposium on Theory of Computing*, pages 373–382. ACM Press, 1995.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[ID93] C. W. Ip and D. L. Dill. Better verification through symmetry. In Claesen [Cla93].

[Jon83] C. B. Jones. Specification and design of (parallel) programs. In *Proceedings of IFIP'83*, pages 321–332. North-Holland, 1983.

[KC90] R. P. Kurshan and E. M. Clarke, editors. *Proc. 1990 Workshop on Comput.-Aided Verification*, June 1990.

[Kil73] Gary A. Kildall. A unified approach to global program optimization. In *POPL*, pages 194–206, 1973.

[Kle71] S. C. Kleene. *Introduction to Metamathematics.* Wolters-Noordhoff, Groningen, 1971.

[KM89] R. P. Kurshan and K. L. McMillan. A structural induction theorem for processes. In *Proc. 8th Ann. ACM Symp. on Principles of Distributed Computing*, pages 239–247. ACM Press, August 1989.

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

[Krö77] Fred Kröger. Lar: A logic of algorithmic reasoning. *Acta Inf.*, 8:243–266, 1977.

[KU77] John B. Kam and Jeffrey D. Ullman. Monotone data flow analysis frameworks. *Acta Inf.*, 7:305–317, 1977.

[Kur94] Robert P. Kurshan. *Computer-aided verification of coordinating processes: the automata-theoretic approach.* Princeton University Press, 1994.

[Lam80] L. Lamport. "Sometimes" is sometimes "Not Never". In *Ann. ACM Symp. on Principles of Prog. Lang.*, pages 174–185, 1980.

[LC80] L. Liu and E. Clarke. Optimization of busy waiting in conditional critical regions. In *13th Hawaii International Conference on System Sciences*, January 1980.

[Lon93] D. E. Long. *Model Checking, Abstraction, and Compositional Reasoning.* PhD thesis, Carnegie Mellon Univ., 1993.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th Ann. ACM Symp. on Principles of Prog. Lang.*, pages 97–107, January 1985.

[MC79] Carver Mead and Lynn Conway. *Introduction to VLSI Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1979.

[MC81] J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, SE-7, No. 4:417–426, July 1981.

[MC85] B. Mishra and E.M. Clarke. Hierarchical verification of asynchronous circuits using temporal logic. *Theoretical Computer Science*, 38:269–291, 1985.

[MCJ97] W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols, 1997.

[McM93] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic Publishers, 1993.

[McM97] Kenneth L. McMillan. A compositional rule for hardware design refinement. volume 1254 of *LNCS*, pages 24–35, June 1997.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd Int. Joint Conf. on Artificial Intelligence*, pages 481–489, September 1971.

[MO81] Y. Malachi and S. S. Owicki. Temporal specifications of self-timed systems. In H. T. Kung, B. Sproull, and G. Steele, editors, *VLSI Systems and Computations.* Comp. Sci. Press, 1981.

[OG76] Susan Owicki and David Gries. Verifying properties of parallel programs: an axiomatic approach. *Commun. ACM*, 19(5):279–285, 1976.

[OL82] Susan Owicki and Leslie Lamport. Proving liveness properties of concurrent programs. *ACM Trans. Program. Lang. Syst.*, 4(3):455–495, 1982.

[Par74] D. M. R. Park. Finiteness is mu-ineffable. Theory of Computation Report No. 3, 1974.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *5th GI-Conference on Theoretical Computer Science*, pages 167–183. Springer-Verlag, 1981. LNCS 104.

[Pel94] D. Peled. Combining partial order reductions with on-the-fly model-checking. In D. L. Dill, editor, *Proc. 1994 Workshop on Comput.-Aided Verification*, volume 818 of *LNCS*, pages 377–390. Springer-Verlag, June 1994.

[Pix90] C. Pixley. Introduction to a computational theory and implementation of sequential hardware equivalence. In Kurshan and Clarke [KC90], pages 54–64.

[Pnu77] A. Pnueli. The temporal semantics of concurrent programs. In *18th Annual Symposium on Foundations of Computer Science*, 1977.

[Pnu84] A. Pnueli. In transition for global to modular temporal reasoning about programs. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*, volume 13 of *NATO ASI series F*. Springer-Verlag, 1984.

[QS82] J. P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th International Symposium on Programming*, pages 337–350, 1982.

[QS83] J. P. Queille and J. Sifakis. Fairness and related properties in transition systems - a temporal logic to deal with fairness. *Acta Inf.*, 19:195–220, 1983. Presented originally in FOCS 1982.

[Ros94] A. W. Roscoe. Model-checking CSP. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare*, pages 353–378. Prentice-Hall, 1994.

[RS93] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. In *Inf. Comp.*, volume 103(2), pages 299–347, 1993.

[SC86] A. P. Sistla and E. M. Clarke. Complexity of propositional temporal logics. *Journal of the ACM*, 32(3):733–749, July 1986.

[Sch98] David A. Schmidt. Data flow analysis is model checking of abstract interpretations. In *POPL*, pages 38–48, 1998.

[Sif89] J. Sifakis, editor. *Proc. 1989 Int. Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*. Springer-Verlag, June 1989.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math*, 5:285–309, 1955.

[TO80] Richard N. Taylor and Leon J. Osterweil. Anomaly detection in concurrent software by static data flow analysis. *IEEE Trans. Software Eng.*, 6(3):265–278, 1980.

[Val90] A. Valmari. A stubborn attack on the state explosion problem. In Kurshan and Clarke [KC90].

[Vit88] Paul M. B. Vitanyi. "andrei nikolaevich kolmogorov". 1:3–18, 1988.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Ann. Symp. on Logic in Comput. Sci.* IEEE Comp. Soc. Press, June 1986.

[WL89] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In Sifakis [Sif89].