The Birth of Model Checking

Edmund M. Clarke
Department of Computer Science
Carnegie Mellon University

Quote For The Day

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.)

Quote from Clarke & Emerson 81

"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)".

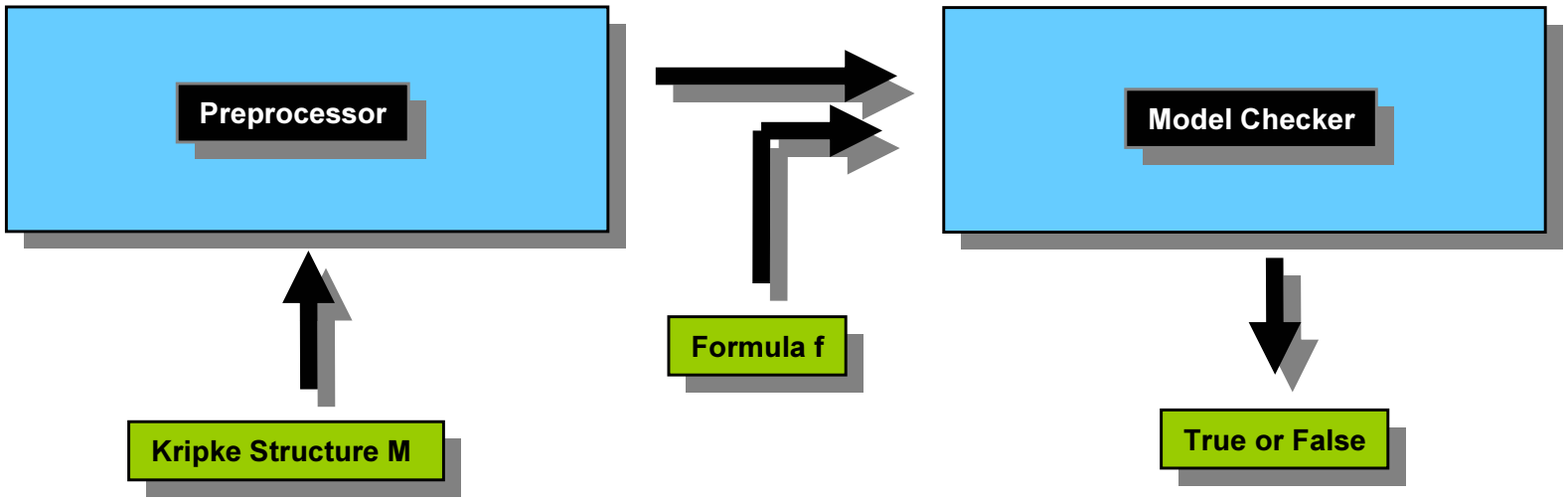
The Model Checking Problem

The Model Checking Problem (CE81):

Let M be a Kripke structure (i.e., state-transition graph).

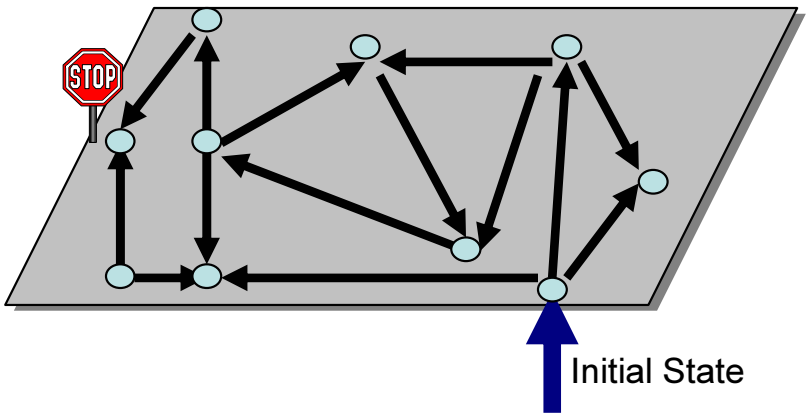
Let f be a formula of temporal logic (i.e., the specification).

Find all states s of M such that $M, s \models f$.



Advantages of Model Checking

- No proofs!!
- Fast (compared to other rigorous methods such)
- Diagnostic counterexamples
- No problem with partial specifications
- Logics can easily express man concurrency properties



Safety Property:
bad state STOP unreachable

Counterexample

Main Disadvantages

- Proving a program helps you understand it.
  Bogus!
- Temporal logic specifications are ugly.
  Depends on who is writing them.
- Writing specifications is hard.
  True, but perhaps partially a matter of education.
- State explosion is a major problem.
  Absolutely true, but we are making progress!

Petri Net Tools

Tadeo Murata:

"I started working on Petri nets from mid-1970, and attended the 1st 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 Comany was selling earlier PN drawing tools and may have had a primitive one before 1981."

Kurt 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 and the early 90's."

Bochmann and Protocol Verification

Gregor Bochmann:

For a workshop organized by André 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."

The Importance of Model Checking

Gregor Bochmann continued:

"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)."

Holzmann and 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."

Holzmann (Continued)

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 'sdlvalid' tool for the verification for SDL (the first such system built) in the period 1987-1990.

That work also led to Spin itself, starting in 1989, which generalized the method and allowed correctness properties to be expressed as omega-regular properties (i.e., as Spin never claims).

The first full Spin version dates from 1989. The converter from LTL to never claims was developed by Doron Peled in 1995 and added in Spin version 2, to make it easier for users to express LTL formulae directly."

Holzmann's use of *Model Checking*

Holzmann:

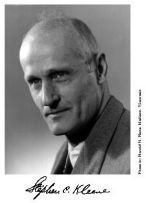
"When do we call "an efficient checker that uses models" a "model checker" though?

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.

*So systems that *only* support basic safety properties (assertions etc.) would not qualify.*

Mu-calculus

- A. Tarski, A Lattice-theoretical fixpoint theorem and its applications, Pacific Journal of Mathematics 5, 285-309, 1955



- S. C. Kleene, Introduction to Meta-Mathematics, 1964. (First Recursion Theorem)

- J. W. de Bakker and D. Scott, A Theory of programs; unpublished notes, IBM Vienna, 1969.

- D.M.R. Park, Fixpoint induction and proofs of program properties, in Machine Intelligence 5, 1970

- D.M.R. Park, Finiteness is Mu-Ineffable, University of Warwick Theory of Computation Report, July 1974.

- E. A. Emerson and E.M. Clarke, Characterizing correctness properties of Parallel programs using fixpoints in LNCS 85, Automata,Languages, and Programming, pp 169-181, Springer 1980.

- D. Kozen, Results on the propositional mu-calculus, Theoretical Computer Science, 27:333-354, 1983.

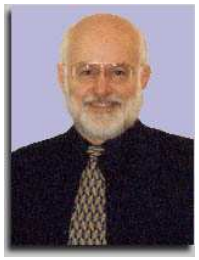
Dataflow Analysis

- G. Killdall: Lattice theoretic approach to iterative data flow Analysis (73)
- Ullmann and students: Monotone data-flow analysis frameworks (76)
- L. Fosdick and L. Osterweil: Data-flow analysis for static error detection (76)
- P. Cousot: Abstract Interpretation and Widening (77, 79)
- D. Schmidt: "Data-flow Analysis is Model Checking of Abstract Interpretations" (98)

Thesis Research

S. Cook, Soundness and Completeness of an Axiom System for Program Verification, (75, 78)

This seminal paper introduced the notion of *Relative Completeness*.



E. Clarke, Completeness and Incompleteness Theorems For Hoare Logics, PhD Thesis, Cornell Computer Science, 76, Advisor: R. Constable

E. Clarke, Programming Language Constructs for which it is impossible to obtain Good Hoare-like Axiom Systems, (77, 79)

E. Clarke, Program Invariants as Fixedpoints, (77, 79)

E. Clarke, S. German, J. Halpern, Effective Axiomatizations of Hoare Logic, 83

E. Clarke, Characterization Problem for Hoare Logics, 84

Thesis Research

Relative Soundness and Completeness Results are really *fixed point theorems*.

I gave a characterization of program invariants as fixed points of functionals obtained from the program text.

- Let **b * A** denote **while b do A.**
- **wp[b * A](Q)** = (: b ∧ Q) ∨ (b ∧ wp[**A**](**wp[b * A](Q)**))
- More complicated for constructs that are not *tail recursive*.

I showed that completeness is logically equivalent to the existence of a fixed point for an appropriate functional.

Soundness follows from the maximality of the fixed point.

For finite interpretations the results give a decision procedure for partial correctness!!!

My Early Research On Concurrency

Fixpoint equations, abstract interpretation, and widening for concurrent programs:

- E. Clarke, "Synthesis of Resource Invariants for Concurrent Programs", 78
- E. Clarke and L. Liu, "Approximate Algorithms for Optimization of busy waiting in Parallel Programs", 79
- L. Liu and E. Clarke, "Optimization of busy waiting in conditional critical regions", 80

Temporal Logic

Temporal logics describe the ordering of events in time without introducing time explicitly.

They were developed by philosophers for investigating how time is used in natural language arguments.

Most have an operator like **G** *f* 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 writes **G** *(: e_1 Ç : e_2)*.

The meaning of a temporal logic formula is determined with respect to a labeled state-transition graph or *Kripke structure*.

- Hughes and Creswell 77

Temporal Logic and Program Verification

Burstall 74, Kroeger 77, and Pnueli 77, all proposed using temporal logic for reasoning about computer programs.

Pnueli 77 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 82 and Owicki and Malachi 81.

Since proofs were constructed by hand, the technique was often difficult to use in practice.

Pnueli 77 and Model Checking

Did Pnueli invent Model Checking in 1977 ???

Section on Finite State Systems most relevant for this meeting.

Theorem 4: *The validity of an arbitrary eventuality* ***G****(A ! **F** B) is decidable for for any finite state system.*

Proof of this theorem uses strongly connected components and is very similar to the technique used for **EG***(P)* in CES 83 / 86.

Pnueli 77 and Model Checking

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

"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 ω-regular language L(W) which describes all those S^{ω} sequences on which W is true..."

"Our decision problem reduces to the question is $L(A_{\Sigma})$ μ L(W), i.e. do all proper execution sequences of Σ satisfy W."

Branching-Time Logics

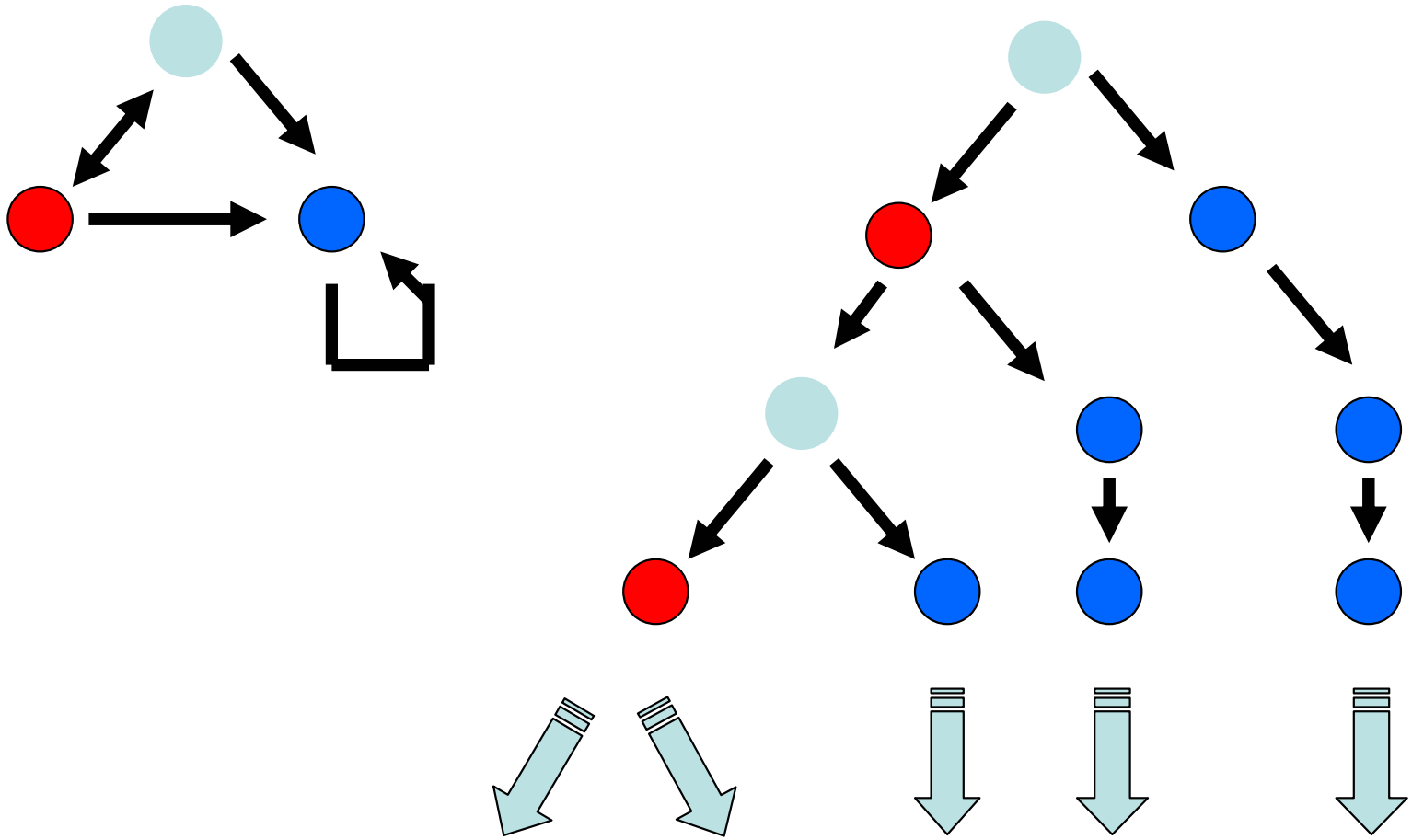
Emerson and Clarke 80 proposed a very general branching-time temporal logic and made the connection with the mu-calculus.

Ben-Ari, Manna, and Pnueli (81 / 83) gave an elegant syntax for a branching time logic called UB.

- EC 80 8 path 9 node p
- BMP 81 AF p

In CE 81 we adopted the UB notation and introduced two versions of the until operator (AU and EU).

Computation Tree Logics



Expressive Power of Temporal Logic

Lamport was the first to investigate the expressive power of various temporal logics for verification.

His 1980 POPL paper discussed two logics: a simple linear-time logic and a simple branching-time logic.

Branching-time logic could not express certain natural fairness properties that can easily expressed in the linear-time logic.

Linear-time logic could not express the possibility of an event occurring at sometime in the future along some computation path.

Technical difficulties with method that Lamport used for his results (somewhat like comparing "apples and oranges").

Emerson and Halpern fixed these problems in an 83 POPL paper

Clarke and Emerson 81

Edmund M. Clarke and E. Allen Emerson, *Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic*. Logics of Programs Workshop, Yorktown Heights, New York, May 1981, LNCS 131. Also in Emerson's Thesis (81).

The temporal logic model checking algorithms of Clarke and Emerson 1980's allowed this type of reasoning to be automated.

Checking that a single model satisfies a formula is much easier than proving the validity of a formula for all models.

The algorithm of Clarke and Emerson for CTL was polynomial in $|M|$ and in $|f|$.

We also showed how fairness could be handled without changing the complexity of the algorithm.

CE 81 Implementation

We got a student (Marshall Brin) to implement the fixpoint algorithm, but the implementation was incorrect (problems with liveness).

We discovered this to our embarrassment when we demonstrated it to Gregor Bochmann!

When I arrived at CMU in the fall of 1982, I implemented the EMC model checker.

Quielle and Sifakis 82

J.P. Queille and J. Sifakis, Specification and Verification of Concurrent Systems in CESAR,

- Technical Report 254 June 1981,
- International Symposium on Programming, Turin, April, 1982
- Springer Lecture Notes in Computer Science 137, published in 1982

Quielle and Sifakis 82 (cont.)

Similarities:

- Branching-time temporal logic related to BMP 80. (**POT** is like **EF** and **INEV** is like our **AF**.)
- Formula Evaluation by computing fixpoints as in CE 81
- CSP and Alternating Bit Example
- Clear distinction between Model (abstraction of program) and Formula to be checked (specification)

Quielle and Sifakis 82 (cont.)

Differences:

- Their logic does not have until **U** operator.
- They do not analyze the complexity of their algorithm.
- They do not have fairness constraints.

QS references Clarke 77 / 80 and Cousot 78 for approximating fixed points of monotonic operators on a lattice.

The EMC Model Checker

Clarke, Emerson, and Sistla (83 / 86) devised 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.

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.

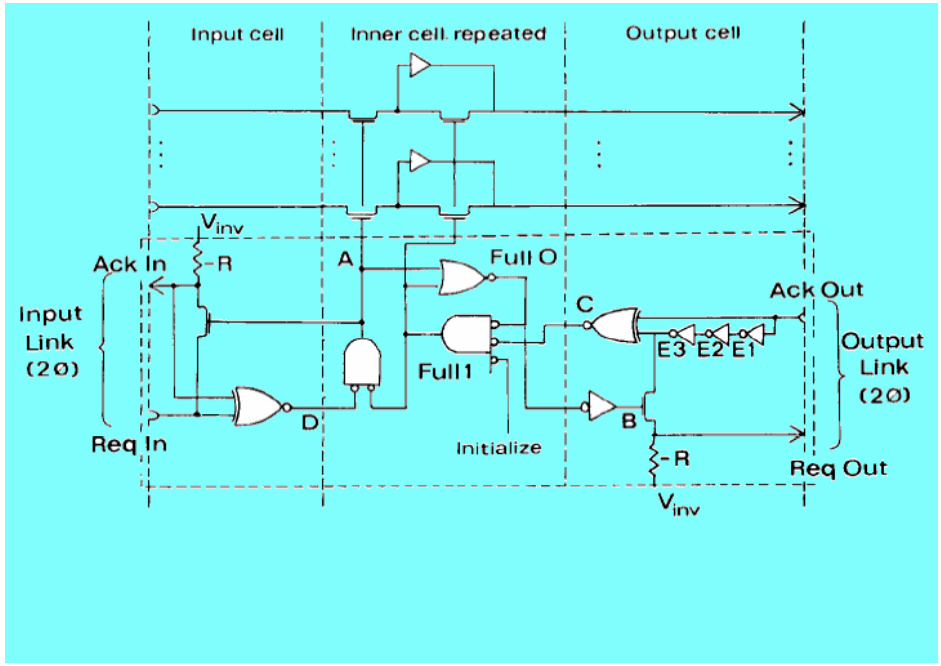
- EMC tool
- Fairness Constraints
- Emptiness of Non-deterministic Buchi Automata

Hardware Verification

- B. Mishra and E. M. Clarke, *Automatic and Hierarchical Verification of Asynchronous Circuits using Temporal Logic*, CMU Tech Report (CMU-CS-83-155) and Theoretical Computer Science 38, 1985, pages 269-291.

First use of Model Checking for Hardware Verification

(found bug in the Sietz FIFO Queue from Mead and Conway, *Introduction to VLSI Systems*).



- Mishra and Clarke 83; Browne, Clarke, and Dill 86; Dill and Clarke 86

Complexity of LTL

Sistla and Clarke (82, 83) analyzed the model checking problem for LTL and showed that the problem was PSPACE-complete.

Pnueli and Lichtenstein (85) 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.

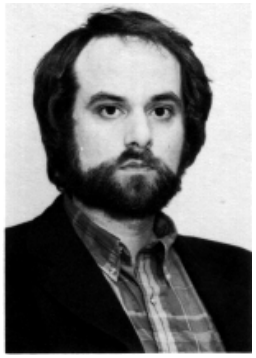
CTL* Model Checking

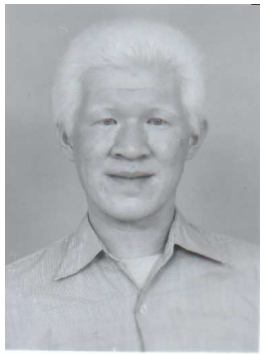
CTL* is a very expressive logic that combines both branching-time and linear-time operators.

Model checking for this logic was first considered in CES 83 / 86 where it was shown to be PSPACE-complete.

Can show that CTL* and LTL model checking have the same complexity in $|M|$ and $|f|$ (Emerson and Lei 85).

Thus, for purposes of model checking, there is no practical complexity advantage to restricting oneself to a linear temporal logic.



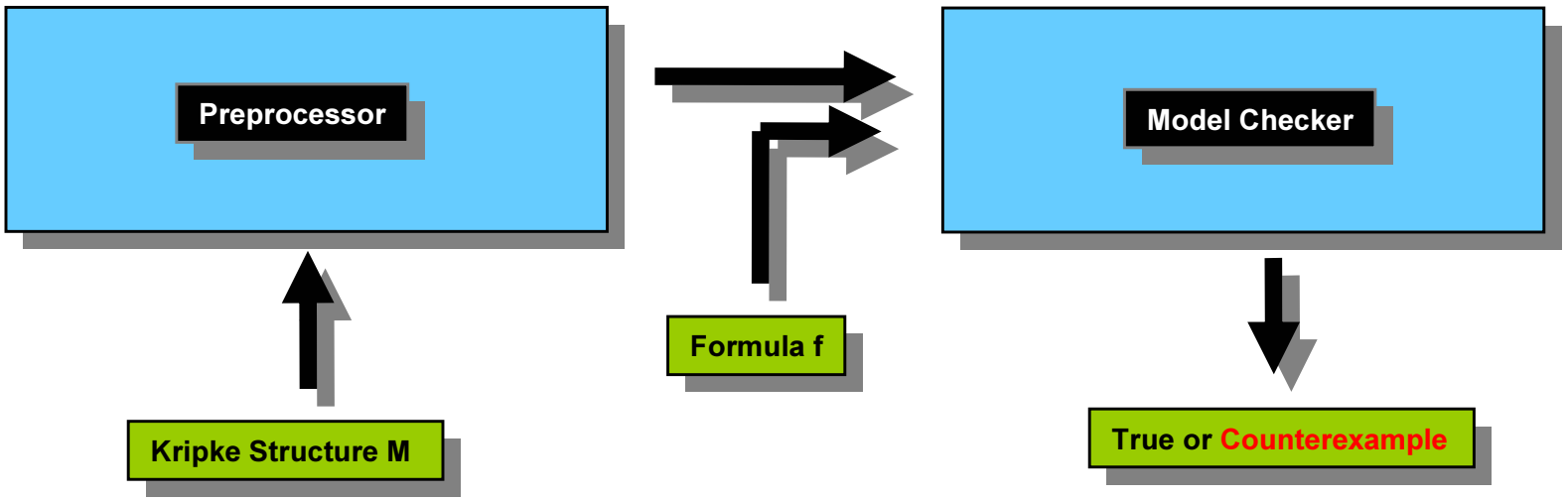


Witnesses and Counterexamples

EMC did not give *counterexamples* for universal CTL properties that were false or *witnesses* for existential properties that were true.

I asked Michael C. Browne (Mike) to add this feature to the MCB model checker in 1984.

It has been an important feature of Model Checkers ever since.



Automata Theoretic Techniques

Some verifiers use automata as both specifications and implementations.

The implementation is checked to see whether its behavior conforms to that of 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.

- Aggarwal, Kurshan, and Sabnani 83; Dill's Thesis 87; Har'El and Kurshan 90

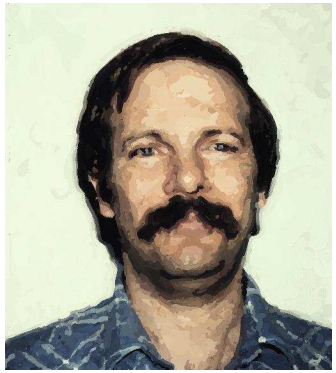
Automata Theoretic MC with LTL

Vardi and Wolper 86 first proposed the use of ω-automata (automata over infinite words) for LTL Model Checking.

They showed how the linear temporal logic model checking could be formulated in terms of language containment.

Most explicit state LTL Model Checkers use a variant of this construction.

Can also be used with Symbolic and Bounded Model Checkers.



Links to Process Algebra

If two finite Kripke Structures can be distinguished by some CTL* formula containing both, then they can be distinguished by a CTL formula.

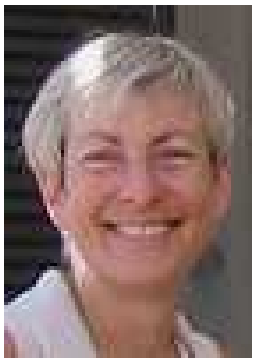
We used a notion of equivalence between Kripke Structures, similar to the notion of bisimulation studied by Milner and Park.

We also considered the case when the next-time operator **X** is disallowed.

The proof in this case requires equivalence with respect to stuttering.

We also gave a polynomial algorithm for determining if two structures are stuttering equivalent.

- Browne, Clarke, and Grumberg 87



Two Big Breakthroughs!

Significant progress was made on the State Explosion Problem around 1990:


- Symbolic Model Checking
  - Coudert, Berthet, and Madre 89
  - Burch, Clarke, McMillan, Dill, and Hwang 90;
  - Ken McMillan's thesis 92

- The Partial Order Reduction
  - Valmari 90
  - Godefroid 90
  - Peled 94



Dealing with Very Complex Systems

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

Four basic techniques are

- Compositional reasoning,
- Abstraction,
- Symmetry reduction, and
- Induction and parameterized verification

Big Events in Model Checking since 1990

- Timed and Hybrid Automata
- Model Checking for Security Protocols
- Bounded Model Checking
- Localization Reduction and CEGAR
- Compositional Model Checking and Learning
- Infinite State Systems (e.g., pushdown systems)

Challenges for the Future

- Software Model Checking, Model Checking and Static Analysis
- Model Checking and Theorem Proving (PVS, STEP, SyMP, Maude)
- 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!!