Automatic Verification of Finite State Concurrent Systems

Edmund M. Clarke, Jr.
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Temporal Logic Model Checking

Specification Language: A propositional temporal logic.

Verification Procedure: Exhaustive search of the state space of the concurrent system to determine truth of specification.

- E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of programs: workshop, Yorktown Heights, NY, May 1981*, volume 131 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.
- J.P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the Fifth International Symposium in Programming*, volume 137 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

Why Model Checking?

Advantages:

- No proofs!!!
- Fast
- Counterexamples
- No problem with partial specifications
- Logics can easily express many concurrency properties

Main Disadvantage: State Explosion Problem

- Too many processes
- Data Paths

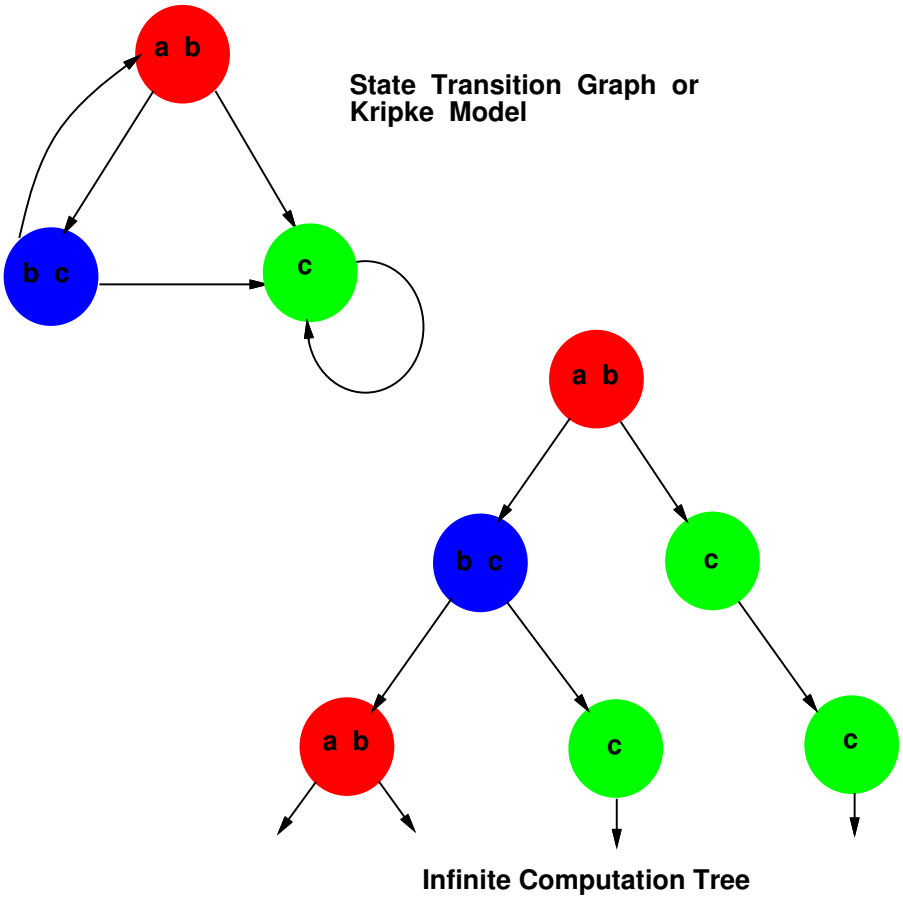
Much progress recently!!

Outline of Talk

---

1. Temporal Logic (CTL*, CTL, and LTL).
2. Model Checking Problem.
3. Some Notable Successes.
4. Symbolic Model Checking with Binary Decision Diagrams.
5. Tomorrow:
   Symbolic Model Checking without Binary Decision Diagrams.
6. Directions for Future Research.

1. Temporal Logic



(Unwind State Graph to obtain Infinite Tree)

Computation Tree Logics

Let M be a Kripke Structure, and let R be the transition relation for M.

A path is an infinite sequence of states $s_0, s_1, \ldots$ such that for every i, $\langle s_i, s_{i+1} \rangle \in R$

1. Path quantifier:
   - **A**—"for every path"
   - **E**—"there exists a path"
2. Temporal Operator:
   - **X**p—p holds next time.
   - **F**p—p holds sometime in the future
   - **G**p—p holds globally in the future
   - p**U**q—p holds until q holds

The Logic CTL*

Two types of formulas in CTL*:

1. A state formula is either
   - a, if a is an atomic proposition, or
   - $\neg f$, $f \vee g$, or $f \wedge g$ where f and g are state formulas, or
   - $\mathbf{E}f$ or $\mathbf{A}f$ where f is a path formula.
2. A path formula is either
   - A state formula, or
   - $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, or $f\mathbf{U}g$ where f and g are path formulas.

The Logics CTL and LTL

In CTL each of the linear-time operators **G**, **F**, **X**, and **U** must be immediately preceded by a path quantifier.

Example: **AG**(**EF** p)

In Linear temporal logic (LTL) formulas have the form **A** f where f is a path formula in which the only state subformulas are atomic propositions.

Example: **A FG** p

The Meaning of Path Quantifiers

Let M be a Kripke structure, s_0 be a state of M, and f be a path formula, then

- $M, s_0 \models \mathbf{E}f$ if and only if there exist a path π starting at s_0, such that $M, \pi \models f$.
- $M, s_0 \models \mathbf{A}f$ if and only if for all paths π starting at s_0, we have $M, \pi \models f$.

Expressive Power

It can be shown that the three logics CTL*, CTL, and LTL have different expressive powers.

For example, there is no CTL formula that is equivalent to the LTL formula $\mathbf{A}(\mathbf{FG}\, p)$.

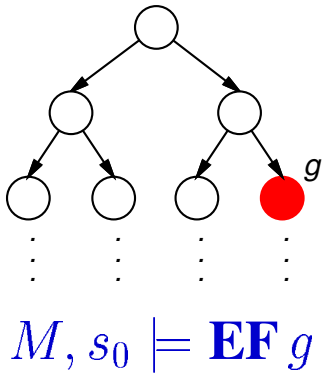
Likewise, there is no LTL formula that is equivalent to the CTL formula $\mathbf{AG}(\mathbf{EF}\, p)$.

The disjunction $\mathbf{A}(\mathbf{FG}\, p) \vee \mathbf{AG}(\mathbf{EF}\, p)$ is a CTL* formula that is not expressible in either CTL or LTL.

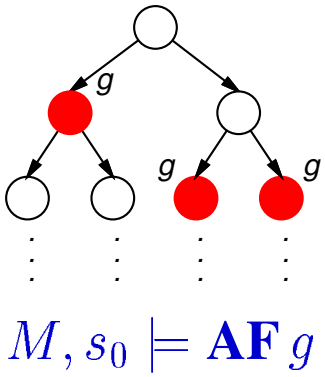
Basic CTL Operators

This lecture will deal primarily with CTL. The four most widely used CTL operators are illustrated below.

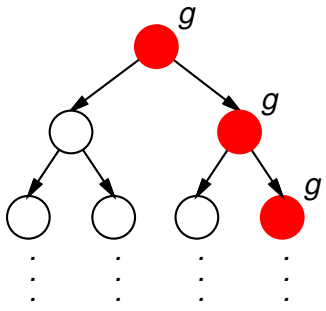
Each computation tree has the state s_0 as its root.



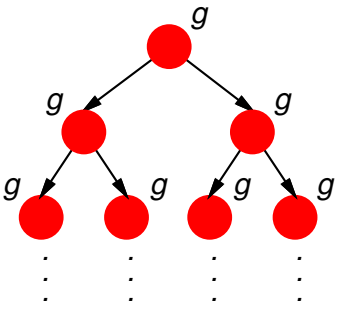
$M, s_0 \models \mathbf{EF}\, g$



$M, s_0 \models \mathbf{AF}\, g$



$M, s_0 \models \mathbf{EG}\, g$



$M, s_0 \models \mathbf{AG}\, g$

Typical CTL* formulas

- **EF**$(Started \wedge \neg Ready)$: it is possible to get to a state where *Started* holds but *Ready* does not hold.
- **AG**$(Req \Rightarrow$ **AF** $Ack)$: if a *Request* occurs, then it will be eventually *Acknowledged*.
- **AG**(**AF** $DeviceEnabled$): *DeviceEnabled* holds infinitely often on every computation path.
- **AG**(**EF** $Restart$): from any state it is possible to get to the *Restart* state.
- **A**(**GF** $Enabled \Rightarrow$ **GF** $Executed$): if a process is infinitely-often *Enabled*, then it is infinitely-often *Executed*.

Note that the first four formulas are CTL formulas. The last is an LTL formula, not expressible in CTL.

2. Model Checking Problem

Let M be the state–transition graph obtained from the concurrent system.

Let f be the specification expressed in temporal logic.

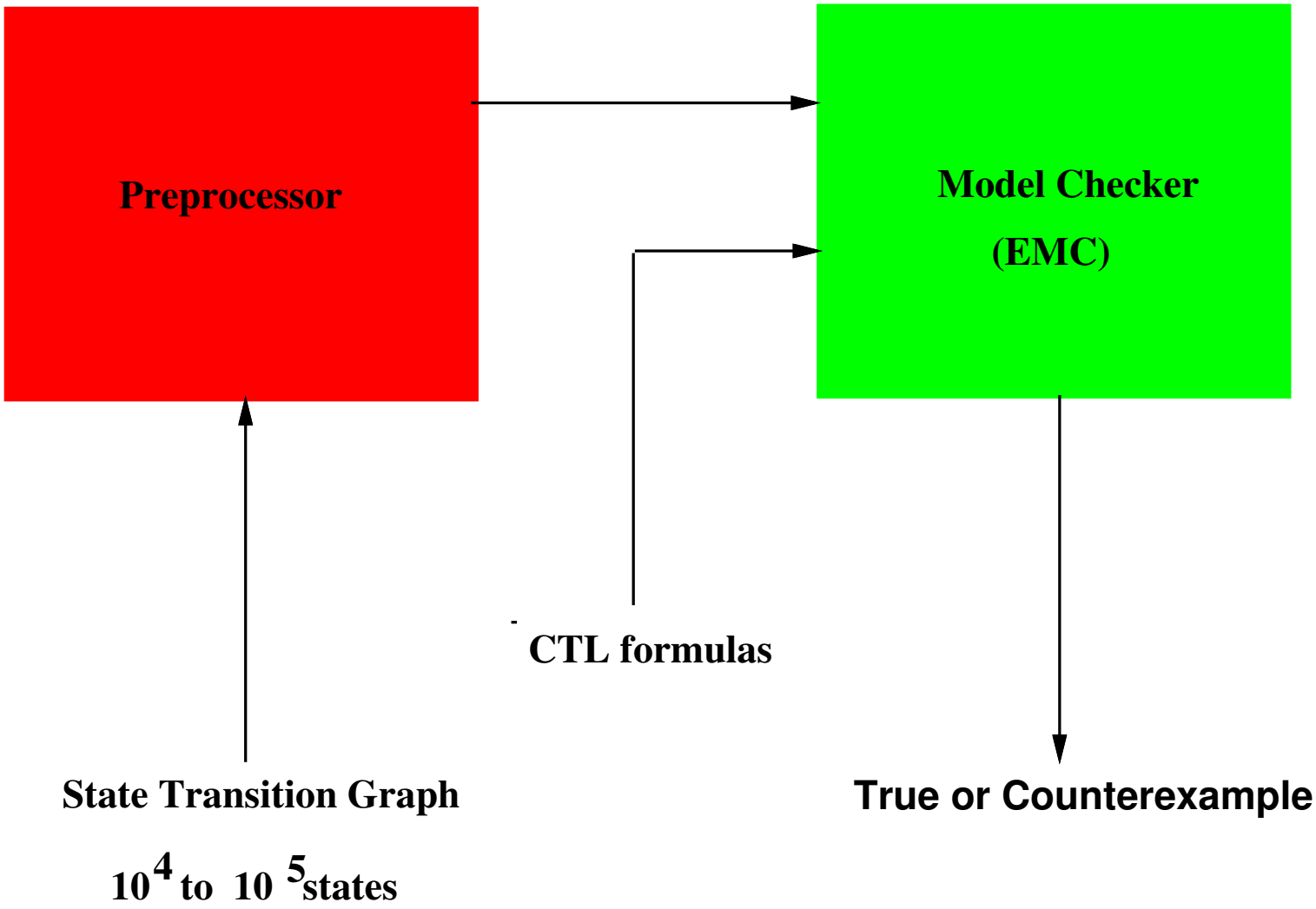
Find all states s of M such that

$$M, s \models f.$$

Efficient model checking algorithms exist for CTL.

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

The EMC System



H. Hiraishi (Kyoto University)

Vectorized version of EMC algorithm on Fujitsu FACOM VP400E Vector Processor using an explicit representation of the state–transition graph.

State Machine size:

- 131,072 states
- 67,108,864 transitions
- 512 transitions from each state on the average.

CTL formula:

- 113 different subformulas.

Time for model checking:

- 225 seconds!!

3. Notable Examples

The following examples illustrate the power of model checking to handle industrial size problems.

They come from many sources, not just my research group.

- Edmund M. Clarke, Jeannette M. Wing, et al. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.

Notable Examples–IEEE Futurebus$^+$

- In 1992 Clarke and his students at CMU used SMV to verify the cache coherence protocol in the IEEE Futurebus+ Standard.
- They constructed a precise model of the protocol and attempted to show that it satisfied a formal specification of cache coherence.
- They found a number of previously undetected errors in the design of the protocol.
- This was the first time that formal methods have been used to find errors in an IEEE standard.
- Although development started in 1988, all previous attempts to validate Futurebus+ were based on informal techniques.

Notable Examples–IEEE SCI

- In 1992 Dill and his students at Stanford used Murφ to verify the cache coherence protocol of the IEEE Scalable Coherent Interface.
- They modeled a typical configuration using the C code in the definition of the SCI standard.
- Since the number of states of the model was very large, they verified only small instances of the system.
- Nevertheless, they found several errors, ranging from uninitialized variables to subtle logical errors.
- The errors also existed in the complete protocol, although it had been extensively discussed, simulated, and even implemented.

Notable Examples–HDLC

- A High-level Data Link Controller (HDLC) was being designed at AT&T in Madrid.
- In 1996 researchers at Bell Labs offered to check some properties of the design. The design was almost finished, so no errors were expected.
- Within five hours, six properties were specified and five were verified, using the FormalCheck verifier.
- The sixth property failed, uncovering a bug that would have reduced throughput or caused lost transmissions.
- The error was corrected in a few minutes and formally verified.

Notable Examples–Analog Circuits

- In 1994, Bosscher, Polak, and Vaandrager won a best-paper award for proving manually the correctness of a control protocol used in Philips stereo components.
- In 1995, Ho and Wong-Toi verified an abstraction of this protocol automatically using HyTech.
- Later in 1995, Daws and Yovine used Kronos to check automatically all the properties stated and hand proved by Bosscher et al.
- In 1996, Bengtsson, et al. model checked the entire protocol. Two years earlier this was considered out of reach for algorithmic methods.

Notable Examples–ISDN/ISUP

- The NewCoRe Project (89-92) was the first full-scale application of formal verification methods in a software project within AT&T.
- Formal modeling and automated verification were applied to the development of the CCITT ISDN User Part Procotol.
- A team of five "verification engineers" formalized and analyzed 145 requirements using a special-purpose model checker.
- A total of 7,500 lines of SDL source code was verified.
- 112 errors were found; about 55% of the original design requirements were logically inconsistent.

Notable Examples–Buildings

- In 1995 the Concurrency Workbench was used to analyze an active structural control system to make buildings more resistant to earthquakes.
- The control system sampled the forces applied to the structure and used hydraulic actuators to exert countervailing forces.
- The first model had more than 10^{19} states and was not directly analyzable. By using semantic minimization it was possible to derive a much smaller model.
- A timing error was discovered that could have caused the controller to worsen, rather than dampen, the vibration experienced during earthquakes.

4. Symbolic Model Checking with BDDs

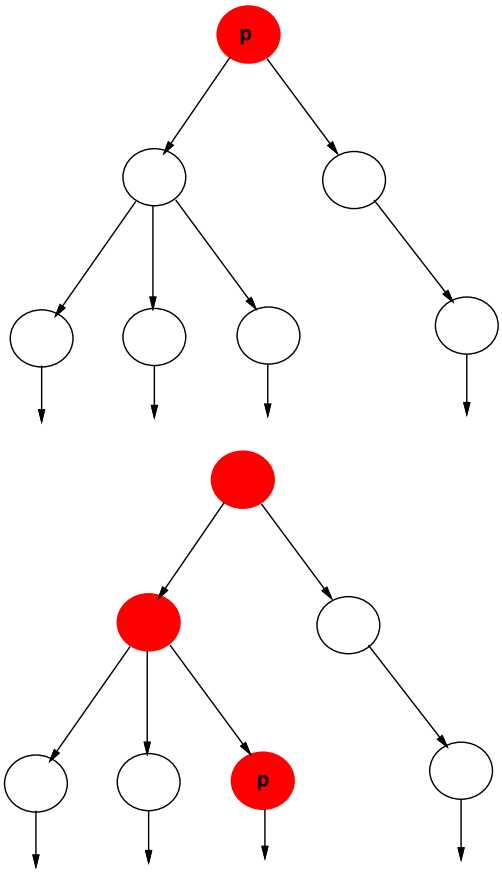
Ken McMillan implemented a version of the CTL model checking algorithm using Binary Decision Diagrams in the fall of 1987.

Now able to handle much larger concurrent systems—some with more than 10^{120} reachable states!!

- J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):pages 142–170, 1992.
- K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

Fixpoint Algorithms

$$\mathbf{EF}\, p = p \vee \mathbf{EX}\, \mathbf{EF}\, p$$



Fixpoint Algorithms (cont.)

Key properties of **EF** p:

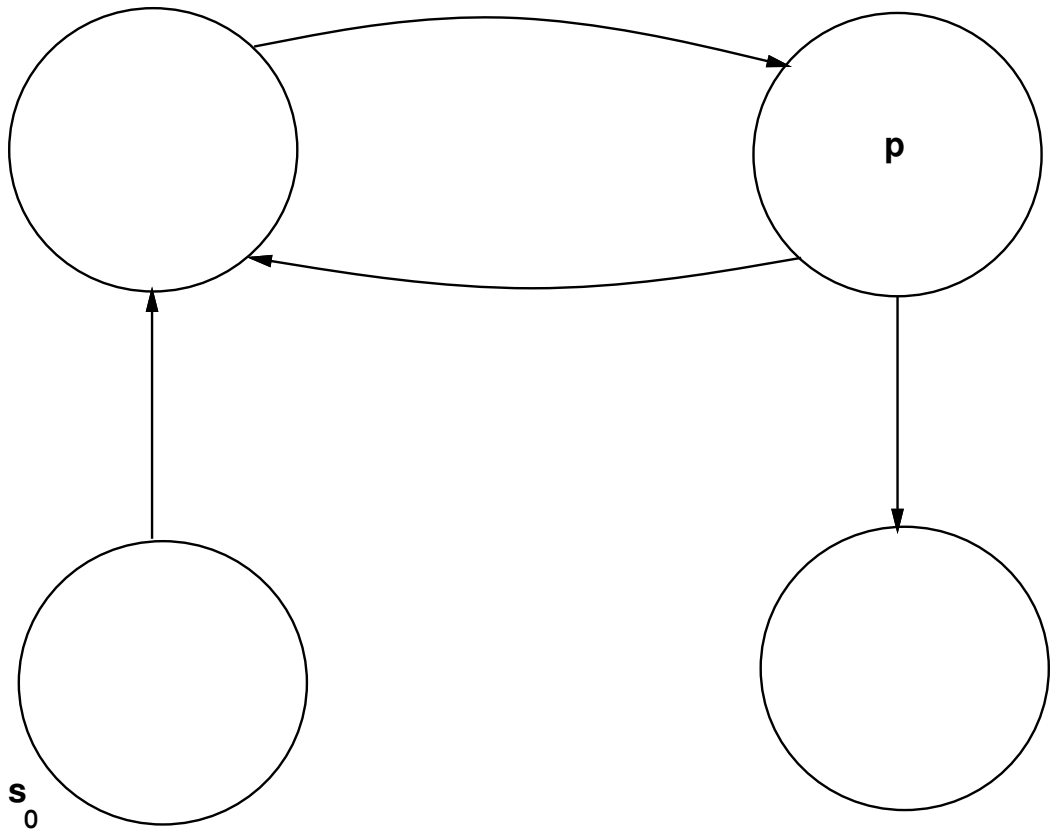
1. **EF** $p = p \vee$ **EX** **EF** p
2. $U = p \vee$ **EX** U implies **EF** $p \subseteq U$

We write **EF** $p =$ **Lfp** $U.p \vee$ **EX** U.

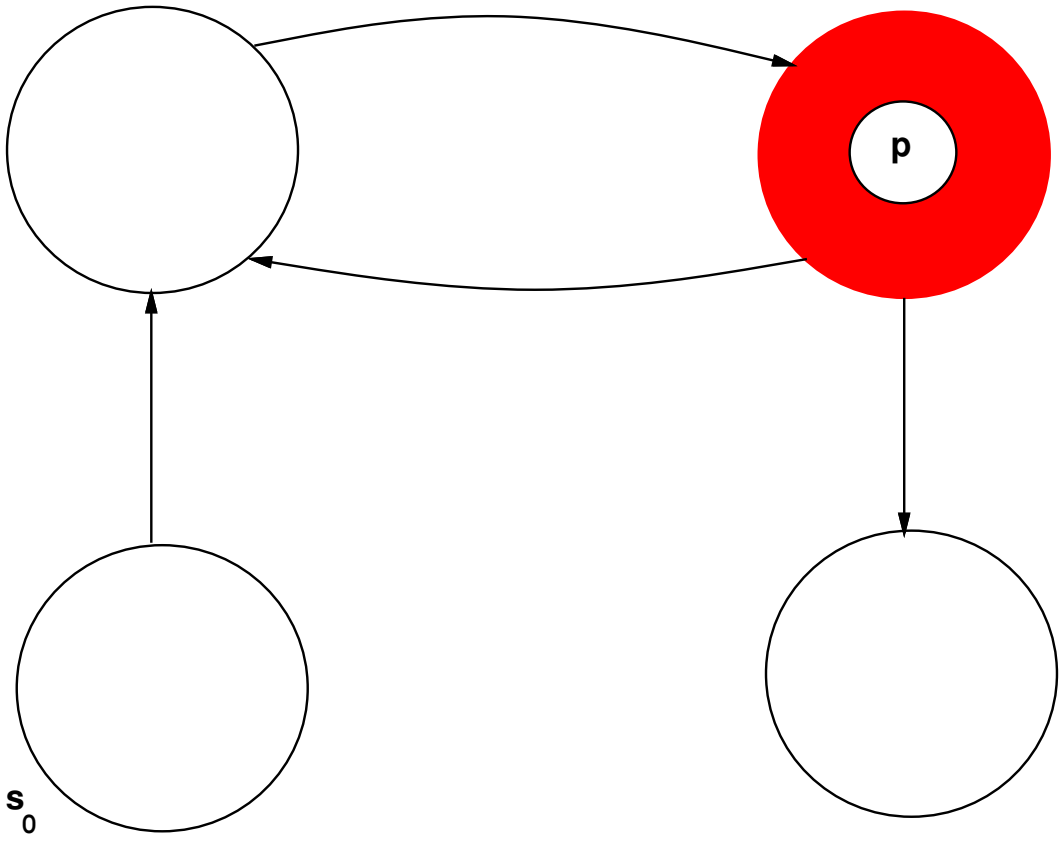
How to compute **EF** p:

$$
\begin{aligned}
U_0 &= \textbf{False} \\
U_1 &= p \vee \textbf{EX}\, U_0 \\
U_2 &= p \vee \textbf{EX}\, U_1 \\
U_3 &= p \vee \textbf{EX}\, U_2 \\
&\vdots
\end{aligned}
$$

$M, s_0 \models \mathbf{EF}\, p$**?**

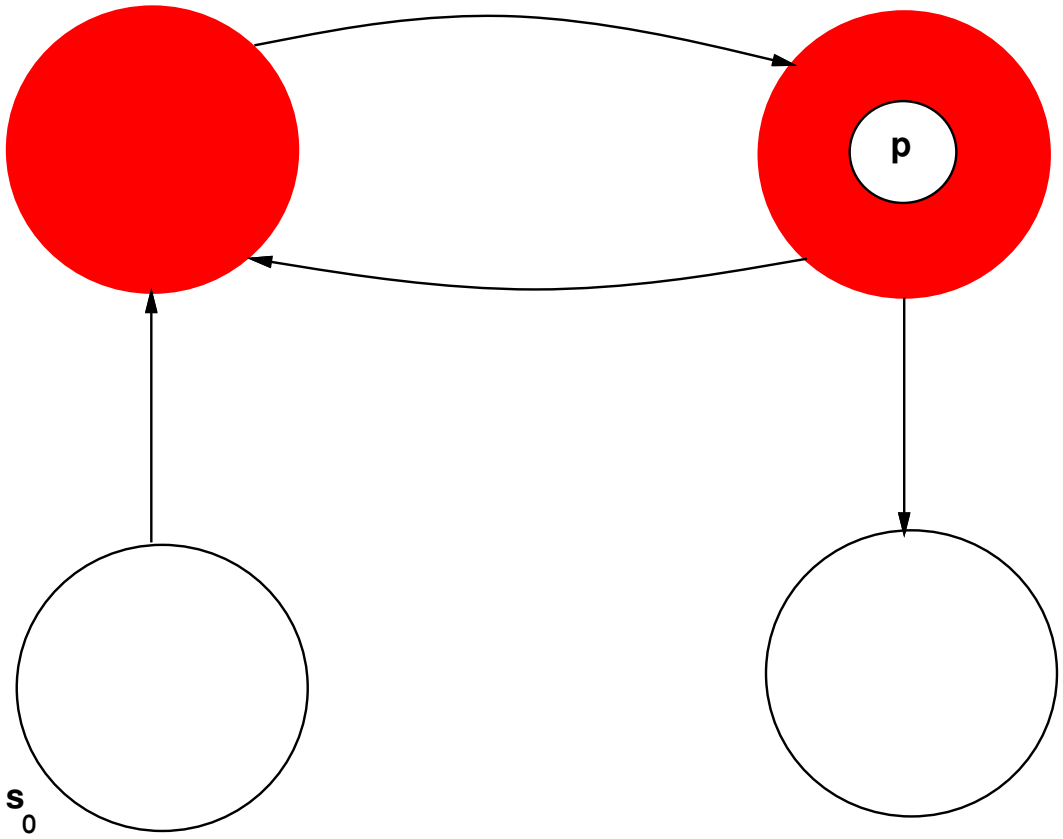


$M, s_0 \models \mathbf{EF}\, p$?



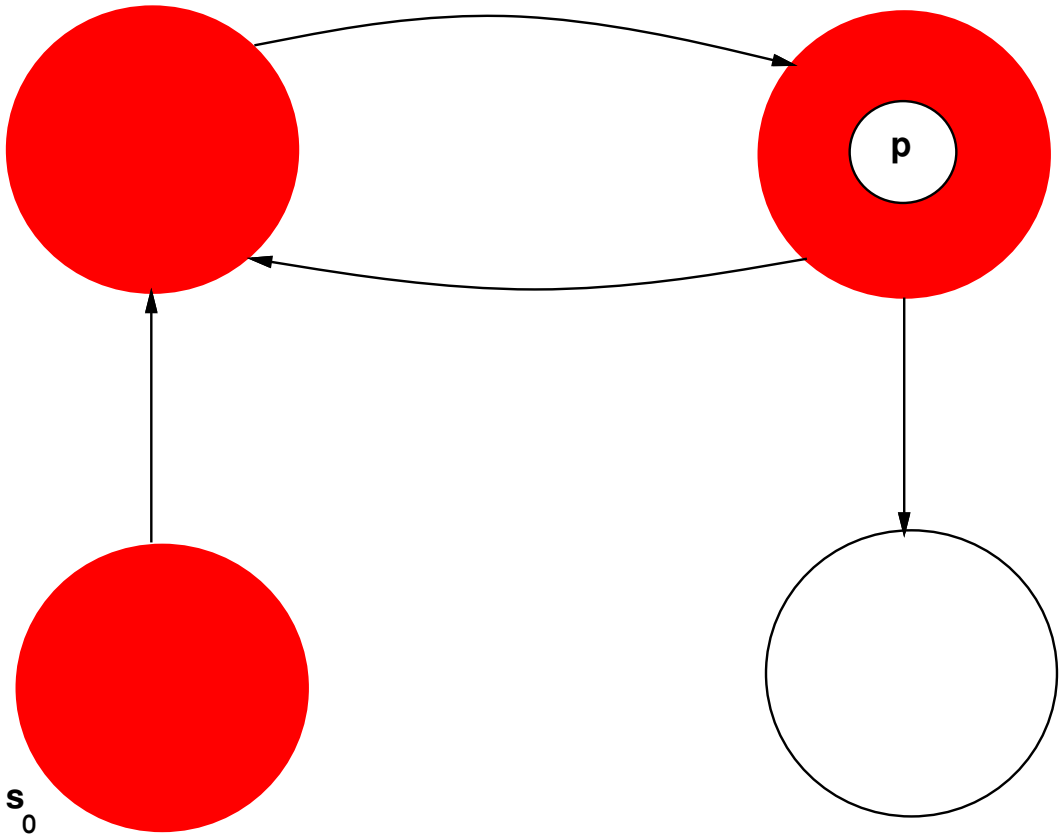
$$U_1 = p \vee \mathbf{EX}\, U_0$$

$M, s_0 \models \mathbf{EF}\, p$?



$$U_2 = p \vee \mathbf{EX}\, U_1$$

$M, s_0 \models \mathbf{EF}\, p$?



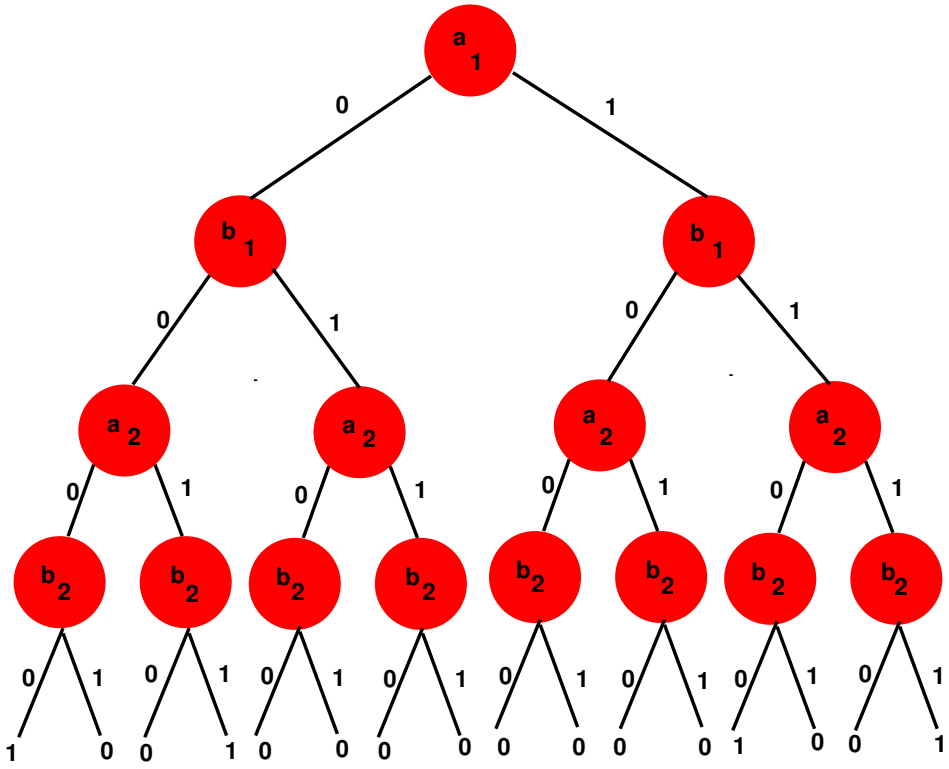
$U_3 = p \vee \mathbf{EX}\, U_2$

Ordered Binary Decision Trees and Diagrams

Ordered Binary Decision Tree for the two-bit comparator, given by the formula

$$f(a_1, a_2, b_1, b_2) = (a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2),$$

is shown in the figure below:



From Binary Decision Trees to Diagrams

An Ordered Binary Decision Diagram (OBDD) is an ordered decision tree where

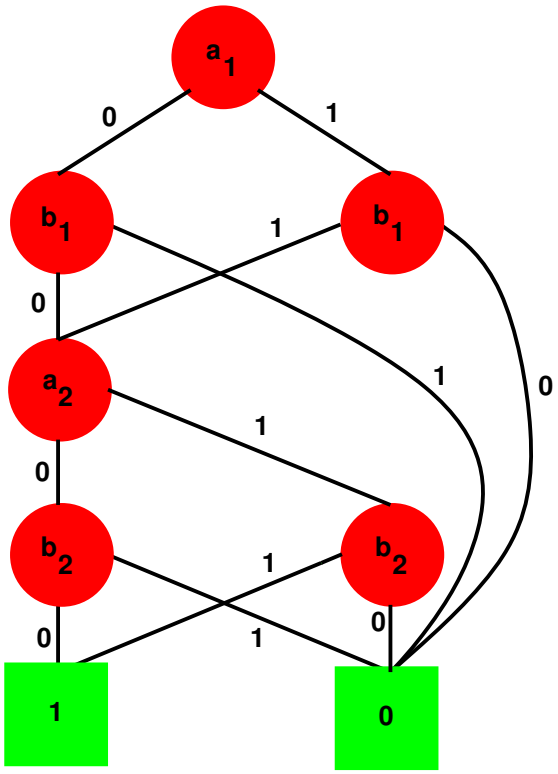
- All isomorphic subtrees are combined, and
- All nodes with isomorphic children are eliminated.

Given a parameter ordering, OBDD is unique up to isomorphism.

- R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

OBDD for Comparator Example

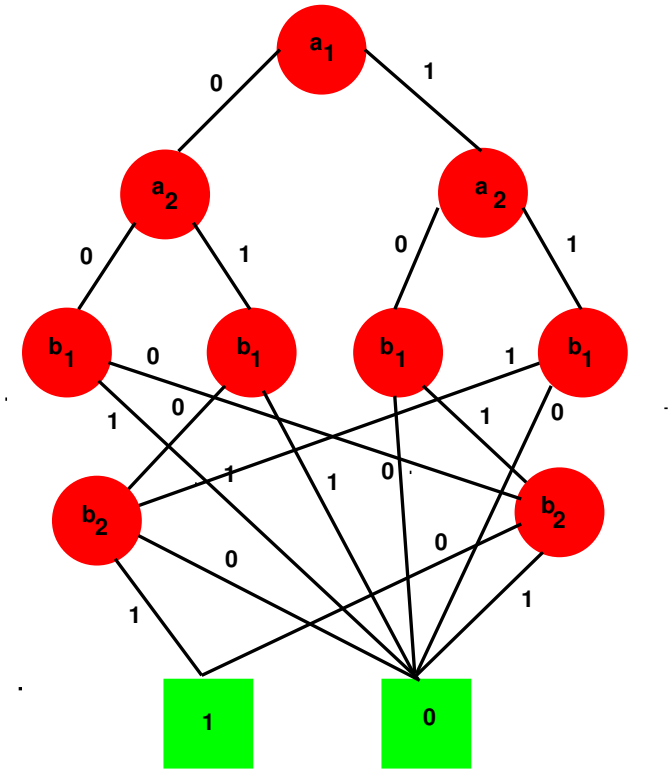
If we use the ordering $a_1 < b_1 < a_2 < b_2$ for the comparator function, we obtain the OBDD below:



Variable Ordering Problem

The size of an OBDD depends critically on the variable ordering.

If we use the ordering $a_1 < a_2 < b_1 < b_2$ for the comparator function, we get the OBDD below:



Variable Ordering Problem (Cont.)

For an n-bit comparator:

- if we use the ordering $a_1 < b_1 < \ldots < a_n < b_n$, the number of vertices will be $3n + 2$.
- if we use the ordering $a_1 < \ldots < a_n < b_1 \ldots < b_n$, the number of vertices is $3 \cdot 2^n - 1$.

In general, finding an optimal ordering is known to be NP-complete.

Moreover, there are boolean functions that have exponential size OBDDs for any variable ordering.

An example is the middle output (n^{th} output) of a combinational circuit to multiply two n bit integers.

Logical operations on OBDD's

- Logical negation: $\neg f(a, b, c, d)$

  Replace each leaf by its negation

- Logical conjunction: $f(a, b, c, d) \wedge g(a, b, c, d)$
  - Use Shannon's expansion as follows,

    $$f \cdot g = \bar{a} \cdot (f|_{\bar{a}} \cdot g|_{\bar{a}}) + a \cdot (f|_a \cdot g|_a)$$

    to break problem into two subproblems. Solve subproblems recursively.
  - Always combine isomorphic subtrees and eliminate redundant nodes.
  - Hash table stores previously computed subproblems
  - Number of subproblems bounded by $|f| \cdot |g|$.

Logical operations (cont.)

- Boolean quantification: $\exists a : f(a, b, c, d)$
  - By definition,

$$\exists a : f = f|_{\bar{a}} \vee f|_{a}$$

  - $f(a, b, c, d)|_{\bar{a}}$: replace all a nodes by left sub-tree.
  - $f(a, b, c, d)|_{a}$: replace all a nodes by right sub-tree.

Using the above operations, we can build up OBDD's for complex boolean functions from simpler ones.

Symbolic Model Checking Algorithm

How to represent state-transition graphs with Ordered Binary Decision Diagrams:

Assume that system behavior is determined by n boolean state variables $v_1, v_2, \ldots, v_n$.

The Transition relation N will be given as a boolean formula in terms of the state variables:

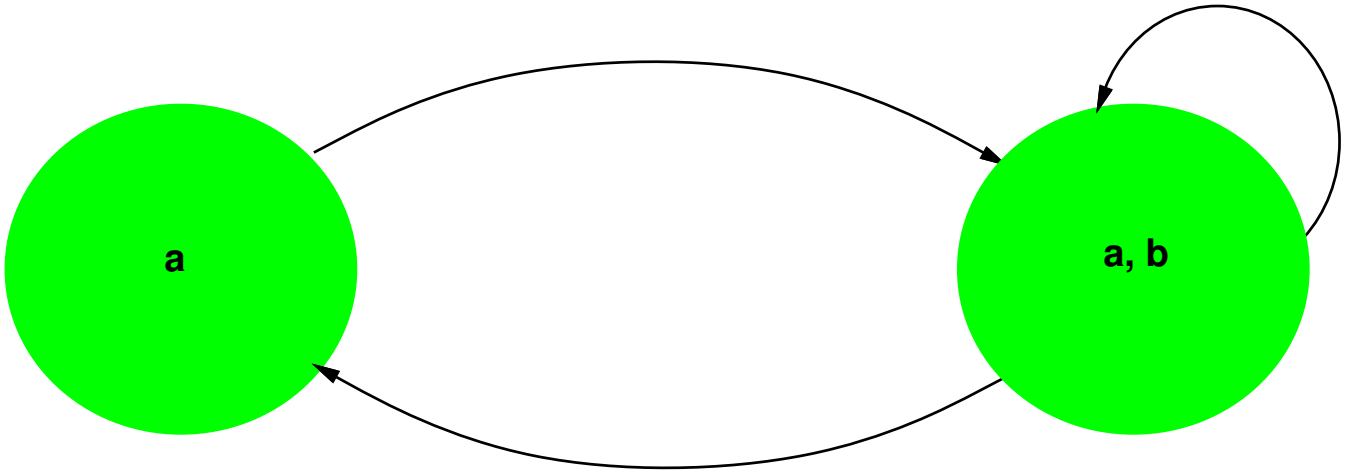
$$N(v_1, \ldots, v_n, v'_1, \ldots, v'_n)$$

where $v_1, \ldots v_n$ represents the current state and $v'_1, \ldots, v'_n$ represents the next state.

Now convert N to a OBDD!!

Symbolic Model Checking (cont.)

Representing transition relations symbolically:



Boolean formula for transition relation:

$$(a \wedge \neg b \wedge a' \wedge b') \vee (a \wedge b \wedge a' \wedge b') \vee (a \wedge b \wedge a' \wedge \neg b')$$

Now, represent as an OBDD!

Symbolic Model Checking (cont.)

Consider $f = \mathbf{EX}\, p$.

Now, introduce state variables and transition relation:

$$f(\bar{v}) = \exists \bar{v}'[N(\bar{v}, \bar{v}') \wedge p(\bar{v}')]$$

Compute OBDD for relational product on right side of formula.

Symbolic Model Checking (cont.)

How to evaluate fixpoint formulas using OBDDs:

$$\mathbf{EF}\, p = \mathbf{Lfp}\; U.\, p \vee \mathbf{EX}\, U$$

Introduce state variables:

$$\mathbf{EF}\, p = \mathbf{Lfp}\; U.\, p(\bar{v}) \vee \exists \bar{v}'\, [N(\bar{v}, \bar{v}') \wedge U(\bar{v}')]$$

Now, compute the sequence

$$U_0(\bar{v}), U_1(\bar{v}), U_2(\bar{v}), \ldots$$

until convergence.

Convergence can be detected since the sets of states $U_i(\bar{v})$ are represented as OBDDs.

Future Research

- Integrate abstraction and compositional reasoning techniques into current verification systems.
- Continue research on the use of symmetry to reason about complex systems.
- Develop methods for verifying parameterized designs. Investigate the use of induction with model checking techniques.
- Develop practical tools for reasoning about realtime and hybrid systems
- Combine model checking techniques with deductive approaches to verification so that control and data can both be handled.
- Develop tool interfaces that are suitable for system designers.