Lecture1: Symbolic Model Checking with BDDs

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:

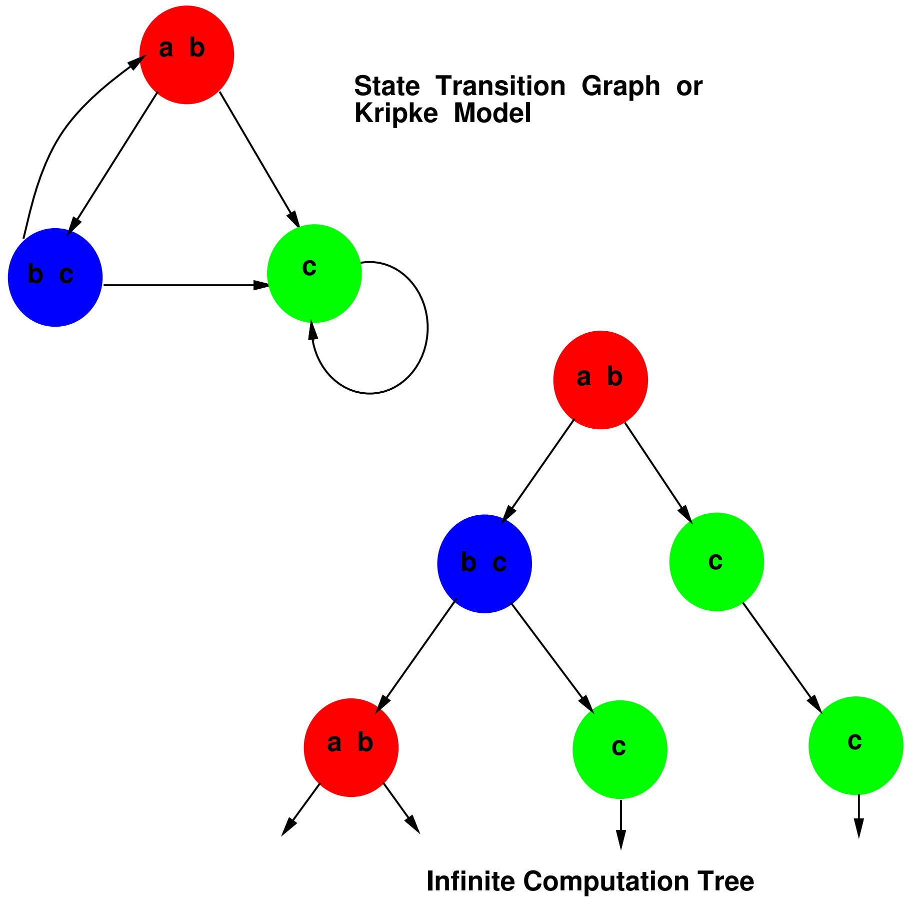
- No proofs!!!
- Fast
- Counterexamples
- No problem with partial specifications
- Logics can easily express many concurrency properties

Main Disadvantage: State Explosion Problem

- Too many processes
- In digital hardware terms: too many latches

Much progress recently!!

Temporal Logic



(Unwind State Graph to obtain Infinite Tree)

Computation Tree Logics

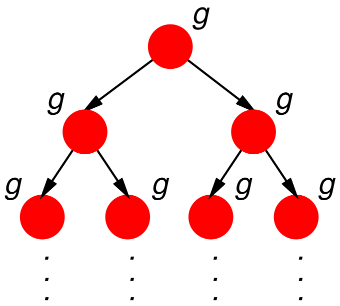
Formulas are constructed from path quantifiers and temporal operators:

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

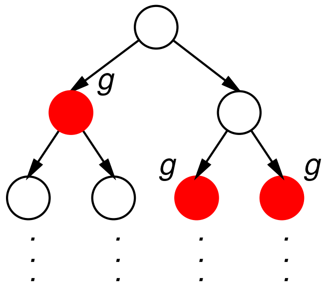
The Logic CTL

In CTL each temporal operator must be immediately preceeded by a path quantifier.

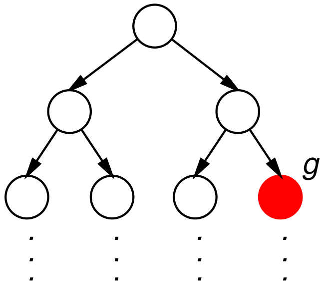
The four most widely used CTL operators are illustrated below. Each computation tree has initial state s_0 as its root.



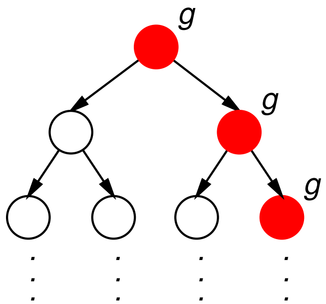
$M, s_0 \models \mathbf{AG}\, g$



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



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



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

Typical CTL Formulas

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

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

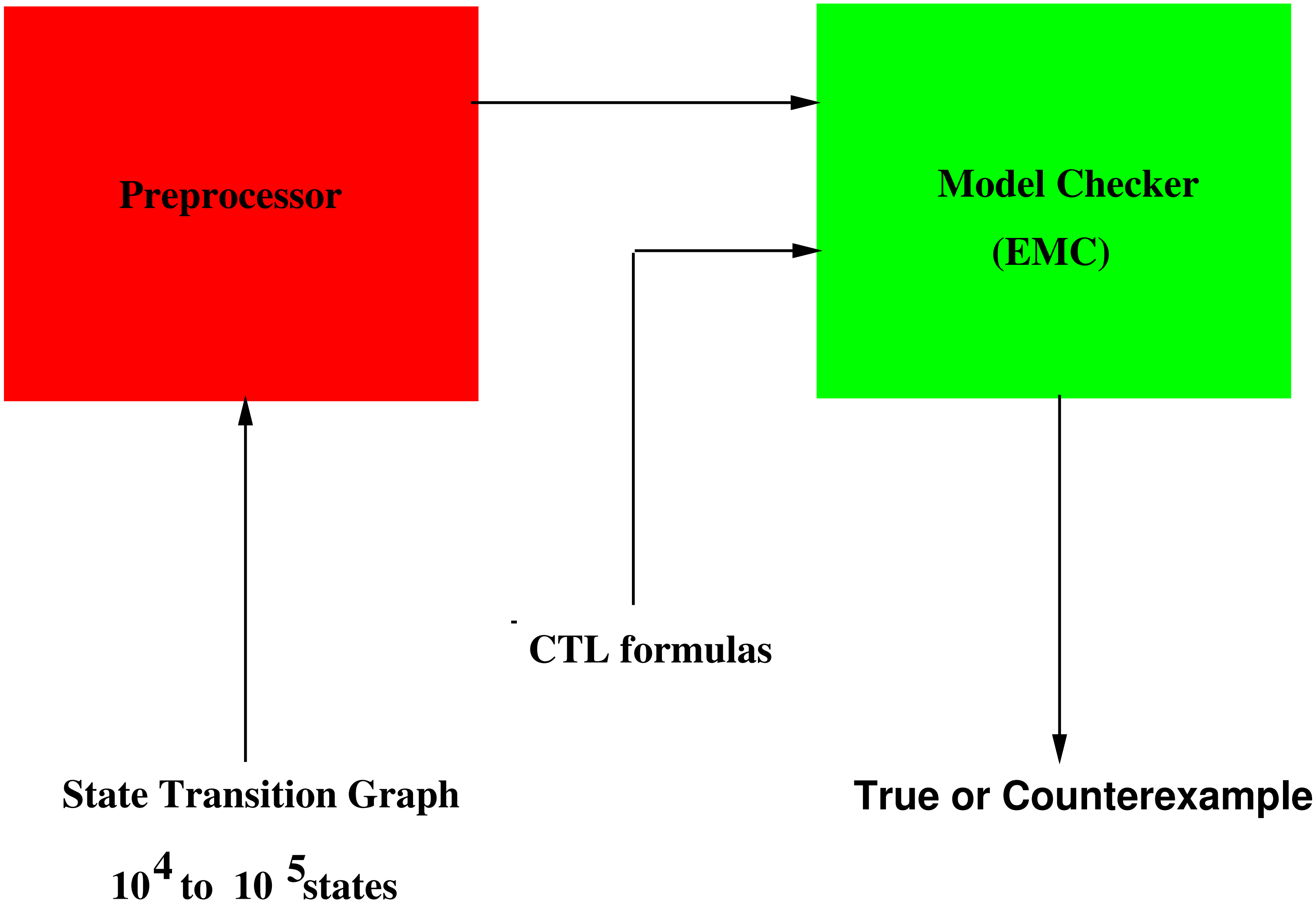
$$M, s \models f$$

and check if initial states are among these.

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.

Explicit Traversal



Symbolic Model Checking

Method used by most "industrial strength" model checkers:

- uses boolean encoding for state machine and sets of states.
- can handle much larger designs – hundreds of state variables.
- BDDs traditionally used to represent boolean functions.

Symbolic Model Checking with BDDs

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

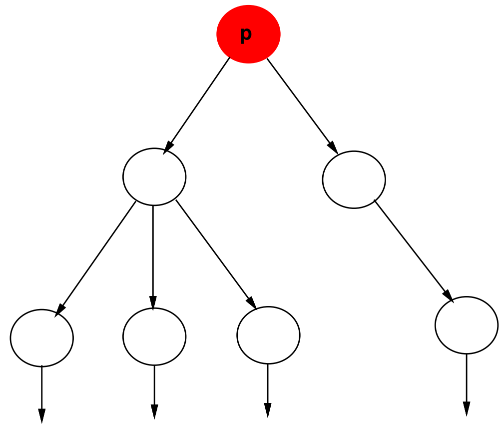
Carl Pixley independently developed a similar algorithm, as did the French researchers, Coudert and Madre.

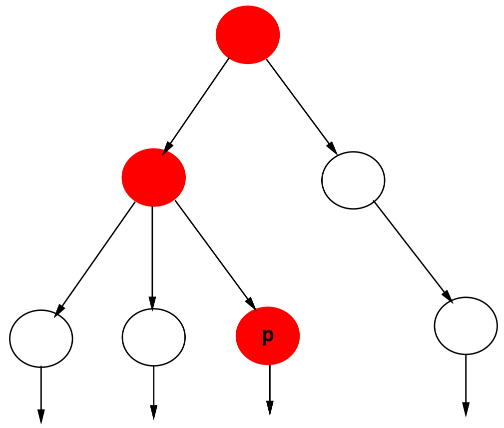
BDDs enabled handling much larger concurrent systems. (usually, an order of magnitude increase in hardware latches!)

- 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 $\mathbf{EF}\, p$:

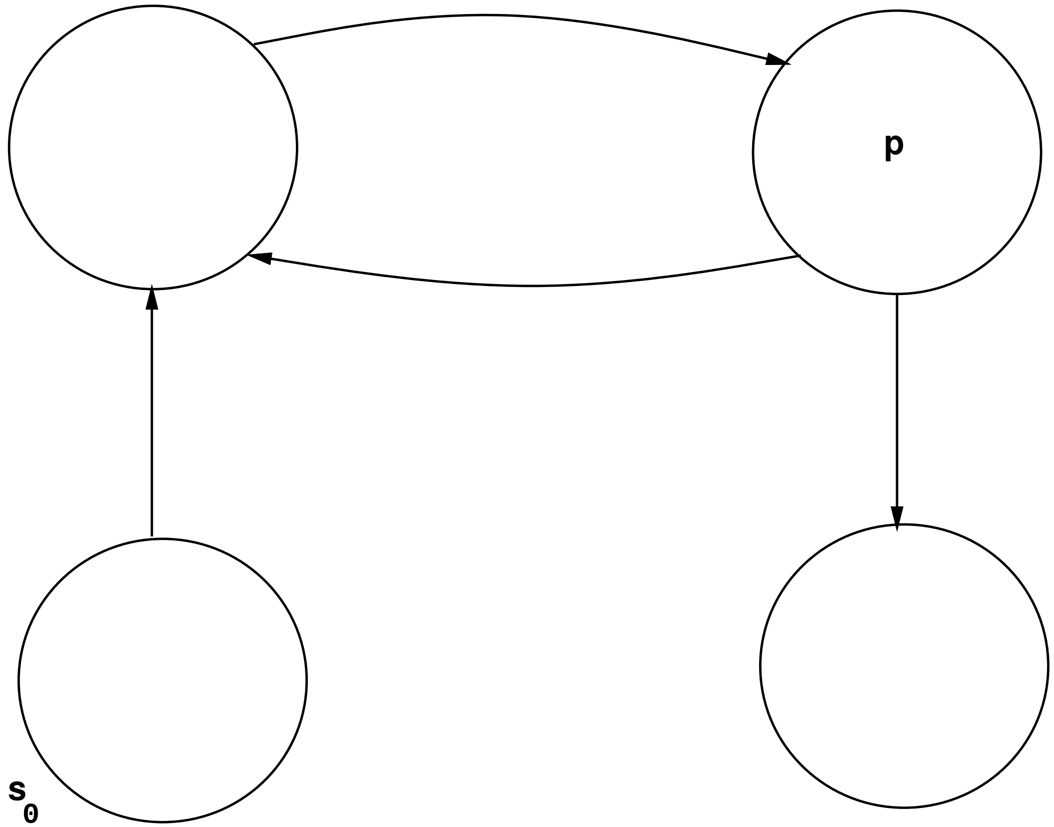
1. $\mathbf{EF}\, p = p \vee \mathbf{EX}\,\mathbf{EF}\, p$
2. $U = p \vee \mathbf{EX}\, U$ implies $\mathbf{EF}\, p \subseteq U$

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

How to compute $\mathbf{EF}\, p$:

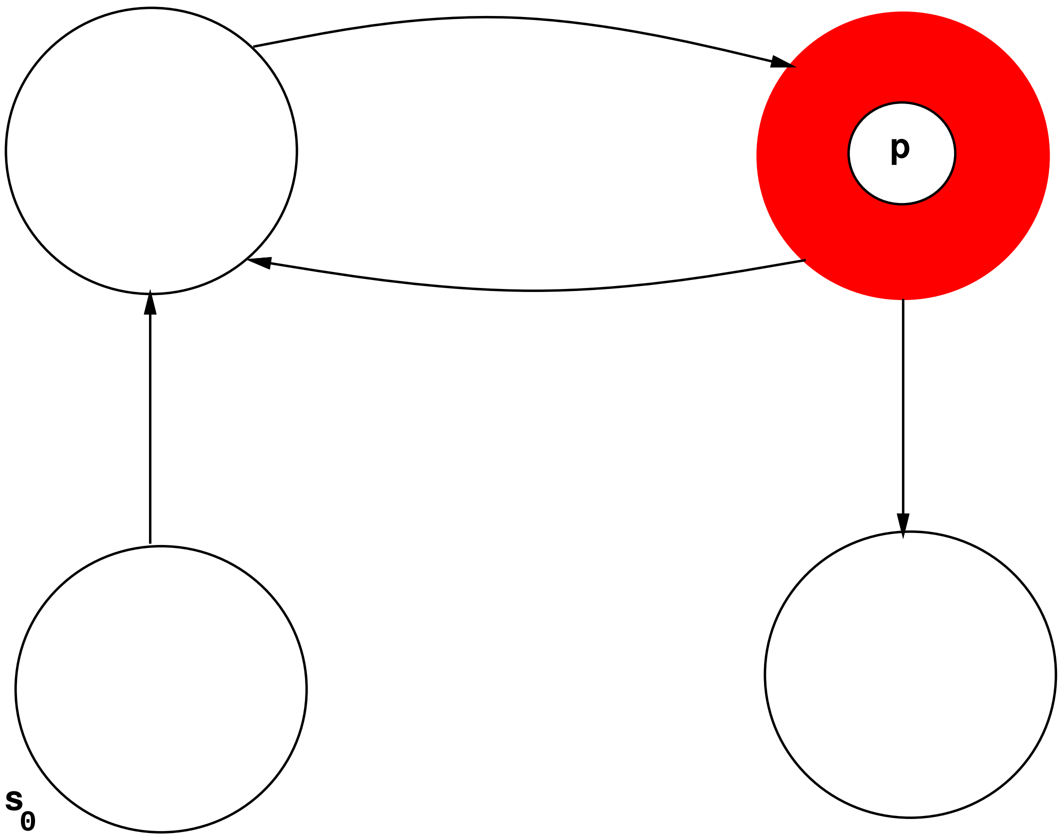
$$
\begin{aligned}
U_0 &= \mathbf{False} \\
U_1 &= p \vee \mathbf{EX}\, U_0 \\
U_2 &= p \vee \mathbf{EX}\, U_1 \\
U_3 &= p \vee \mathbf{EX}\, U_2 \\
&\vdots
\end{aligned}
$$

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



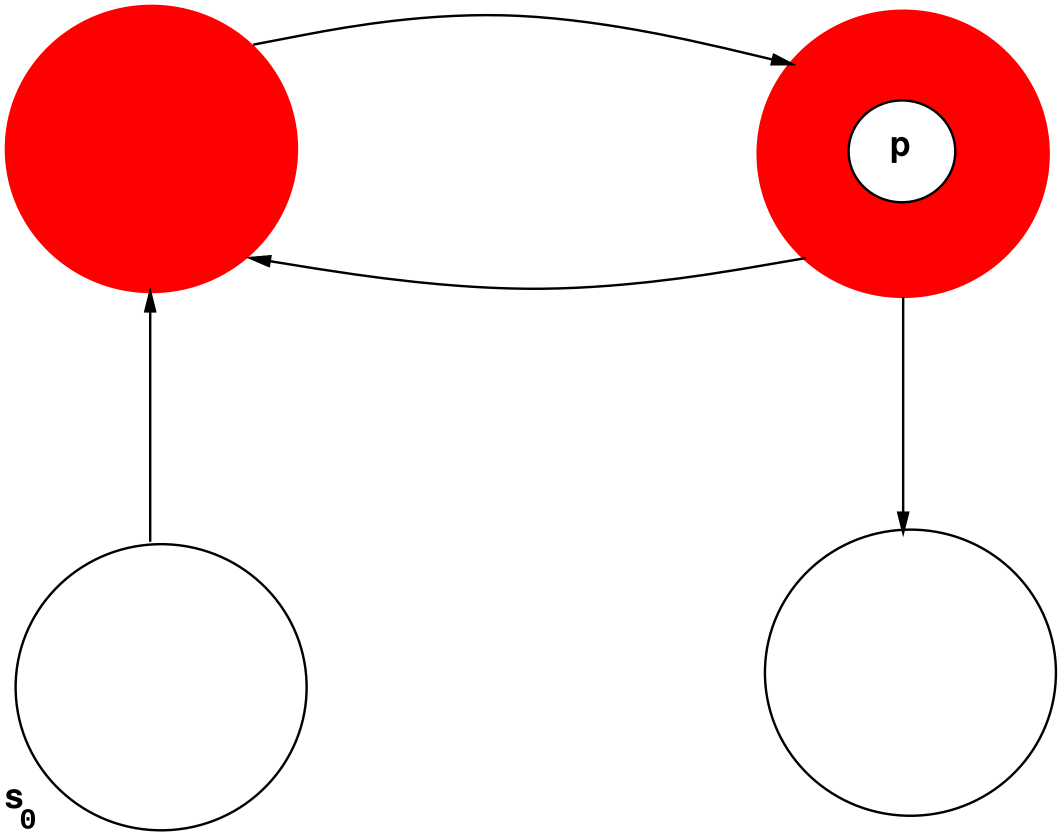
$U_0 = \emptyset$

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



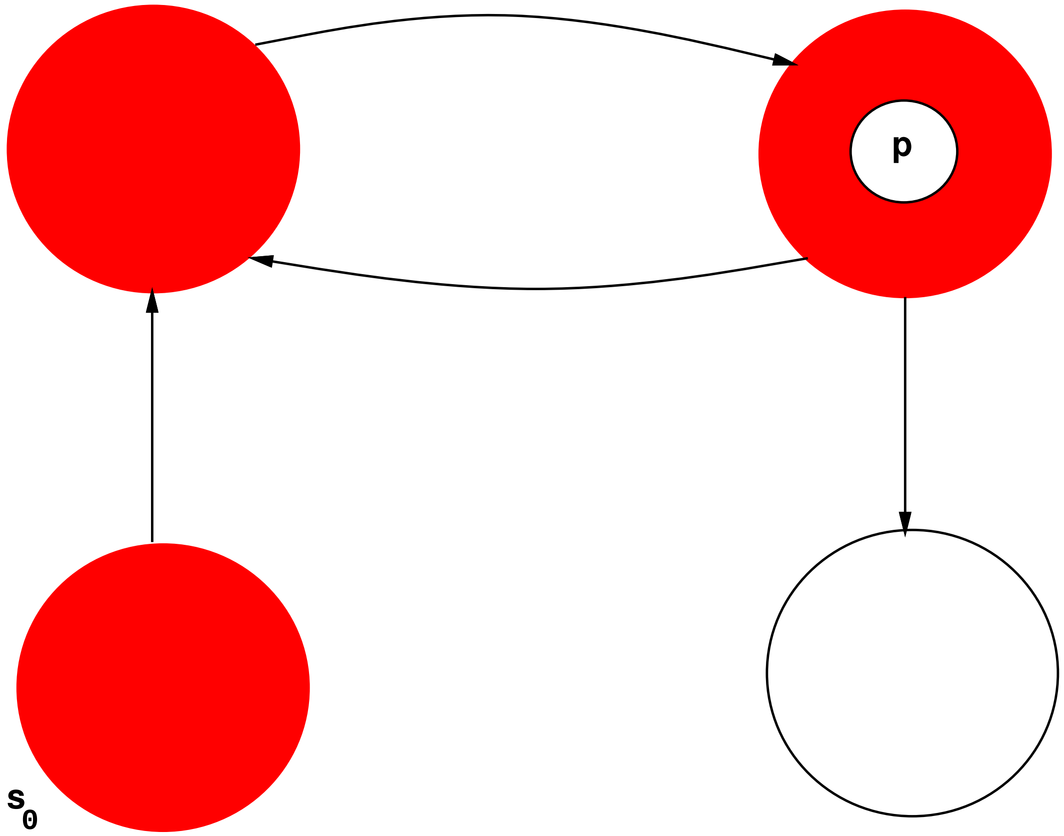
$$U_1 = p \vee \mathbf{EX}\, U_0$$

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



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

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



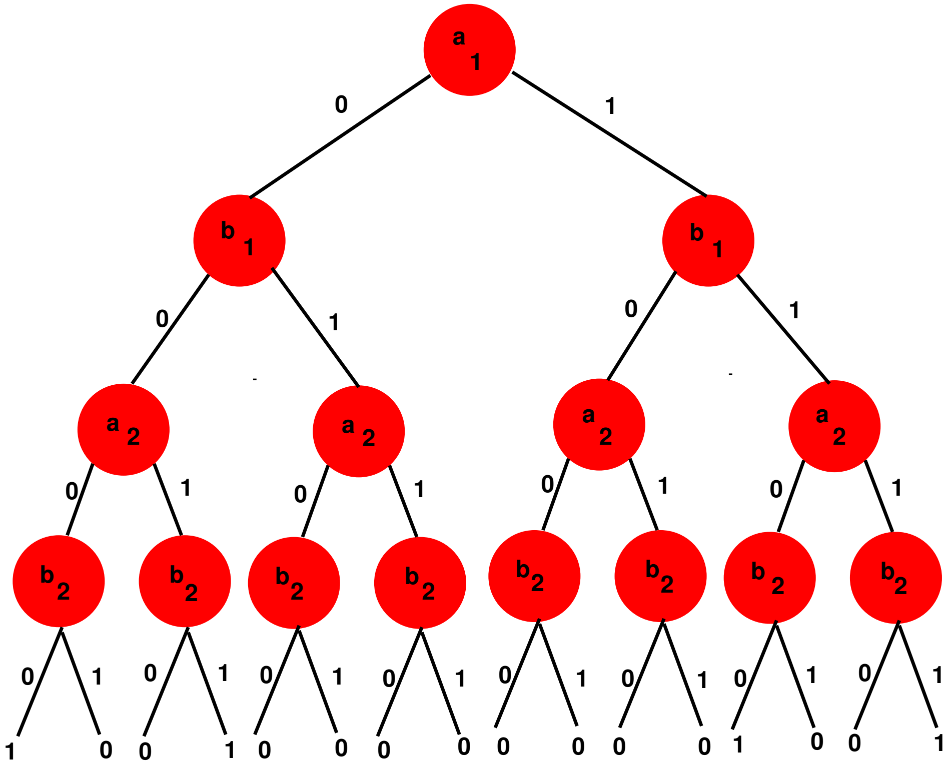
$$U_3 = p \vee \mathbf{EX}\, U_2$$

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

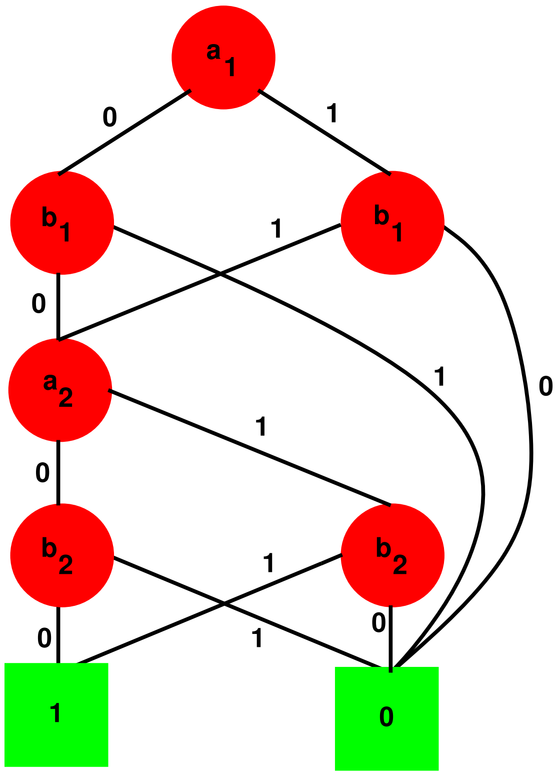
- All isomorphic subtrees are combined, and
- All nodes with isomorphic children are eliminated.

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

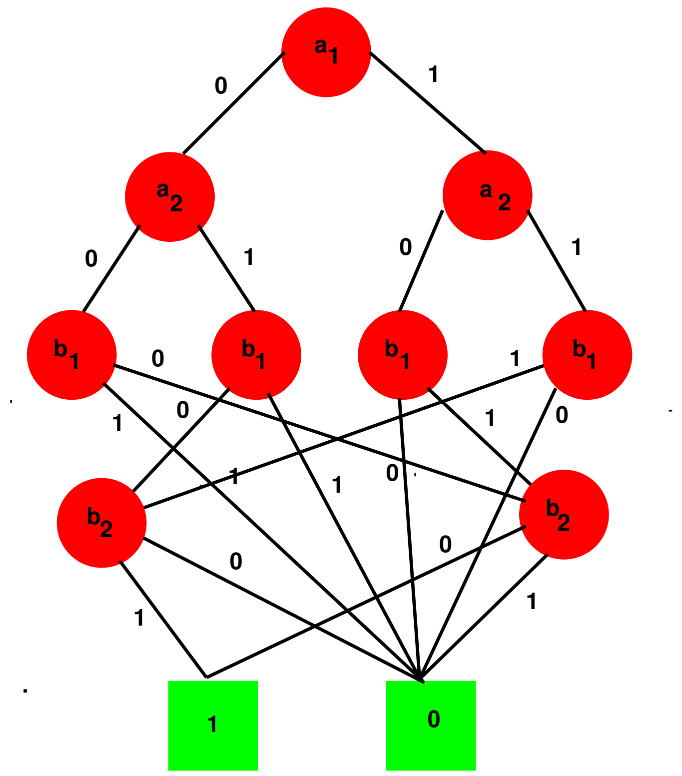
If we use the ordering $a_1 < b_1 < a_2 < b_2$ for the comparator function, we obtain the OBDD below:



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 $3 \cdot 2^n - 1$.

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 T will be given as a boolean formula in terms of the state variables:

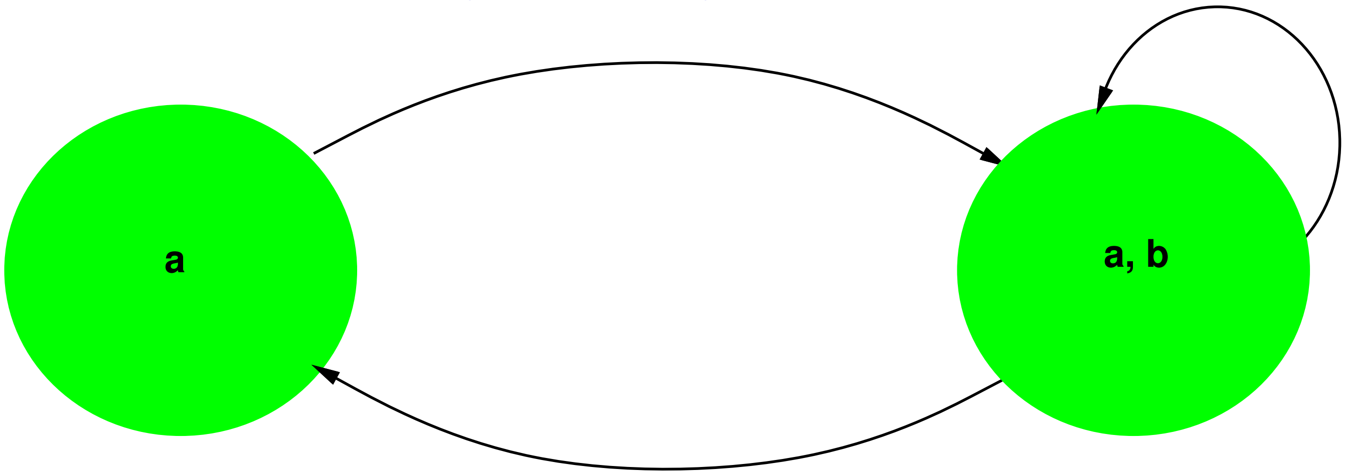
$$T(v_1, \ldots, v_n, v_1', \ldots, v_n')$$

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

Now convert T to a OBDD!!

Symbolic Model Checking (cont.)

Representing transition relations symbolically:



Boolean formula for transition relation:

$$
\begin{array}{lccccccc}
 & (a & \wedge & \neg b & \wedge & a' & \wedge & b') \\
\vee & (a & \wedge & b & \wedge & a' & \wedge & b') \\
\vee & (a & \wedge & b & \wedge & a' & \wedge & \neg b')
\end{array}
$$

Now, represent as an OBDD!

Symbolic Model Checking (cont.)

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

Now, introduce state variables and transition relation:

$$f(\bar{v}) = \exists \bar{v}'[T(\bar{v}, \bar{v}') \wedge p(\bar{v}')]$$

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:

$$\mathbf{EF}\, p = \mathbf{Lfp}\; U.\, p(\bar{v}) \vee \exists \bar{v}' \; [T(\bar{v}, \bar{v}') \wedge U(\bar{v}')]$$

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.

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–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–PowerPC 620 Microprocessor

- Richard Raimi and Jim Lear at Somerset used Motorola's Verdict model checker to debug a hardware laboratory failure.
- Initial silicon of PowerPC 620 microprocessor crashed during boot of an operating system.
- With run time in seconds, Verdict produced example of BIU deadlock causing the failure.
- Paper on this published at 1997 IEEE International Test Conference.

Future Research Directions

Additional work needed on classical model checking:

- Abstraction,
- Compositional Reasoning,
- Symmetry, and
- Parameterized Designs.