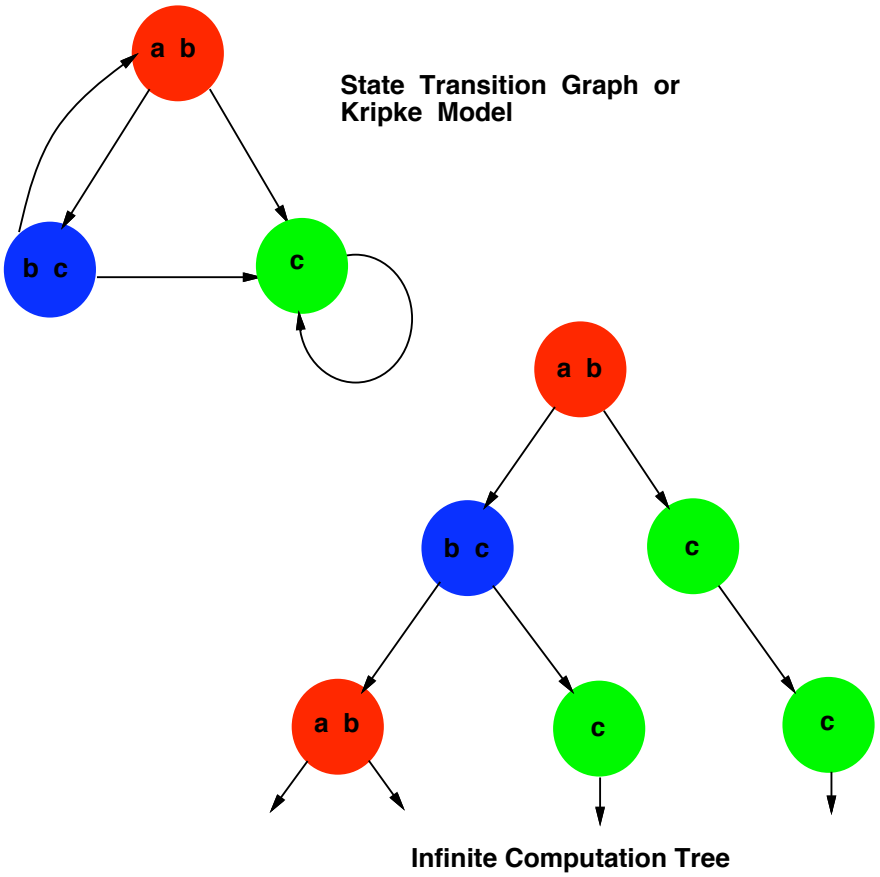
Model of Computation



(Unwind State Graph to obtain Infinite Tree)

Model of Computation (Cont.)

Formally, a Kripke structure is a triple $M = \langle S, R, L \rangle$, where

- S is the set of states,
- $R \subseteq S \times S$ is the transition relation, and
- $L : S \to \mathcal{P}(AP)$ gives the set of atomic propositions true in each state.

We assume that R is total (i.e., for all states $s \in S$ there exists a state $s' \in S$ such that $(s, s') \in R$).

A path in M is an infinite sequence of states, $\pi = s_0, s_1, \ldots$ such that for $i \geq 0$, $(s_i, s_{i+1}) \in R$.

We write π^i to denote the suffix of π starting at s_i.

Unless otherwise stated, all of our results apply only to finite Kripke structures.

Computation Tree Logics

Temporal logics may differ according to how they handle branching in the underlying computation tree.

In a linear temporal logic, operators are provided for describing events along a single computation path.

In a branching-time logic the temporal operators quantify over the paths that are possible from a given state.

The Logic CTL*

The computation tree logic CTL* combines both branching-time and linear-time operators.

In this logic a path quantifier can prefix an assertion composed of arbitrary combinations of the usual linear-time operators.

1. Path quantifier:
   - **A**—"for every path"
   - **E**—"there exists a path"
2. Linear-time operators:
   - **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

Path Formulas and State Formulas

The syntax of state formulas is given by the following rules:

- If $p \in AP$, then p is a state formula.
- If f and g are state formulas, then $\neg f$ and $f \vee g$ are state formulas.
- If f is a path formula, then $\mathbf{E}(f)$ is a state formula.

Two additional rules are needed to specify the syntax of path formulas:

- If f is a state formula, then f is also a path formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $\mathbf{X}\, f$, and $(f\ \mathbf{U}\ g)$ are path formulas.

State Formulas (Cont.)

If f is a state formula, the notation $M, s \models f$ means that f holds at state s in the Kripke structure M.

Assume f_1 and f_2 are state formulas and g is a path formula. The relation $M, s \models f$ is defined inductively as follows:

1. $s \models p \quad \Leftrightarrow p \in L(s)$.
2. $s \models \neg f_1 \quad \Leftrightarrow s \not\models f_1$.
3. $s \models f_1 \vee f_2 \Leftrightarrow s \models f_1$ or $s \models f_2$.
4. $s \models \mathbf{E}(g) \quad \Leftrightarrow$ there exists a path π starting with s such that $\pi \models g$.

Path Formulas (Cont.)

If f is a path formula, $M, \pi \models f$ means that f holds along path π in Kripke structure M.

Assume g_1 and g_2 are path formulas and f is a state formula. The relation $M, \pi \models f$ is defined inductively as follows:

1. $\pi \models f \quad \Leftrightarrow s$ is the first state of π and $s \models f$.
2. $\pi \models \neg g_1 \quad \Leftrightarrow \pi \not\models g_1$.
3. $\pi \models g_1 \vee g_2 \quad \Leftrightarrow \pi \models g_1$ or $\pi \models g_2$.
4. $\pi \models \mathbf{X}\, g_1 \quad \Leftrightarrow \pi^1 \models g_1$.
5. $\pi \models (g_1\ \mathbf{U}\ g_2) \Leftrightarrow$ there exists a $k \geq 0$ such that
   $\pi^k \models g_2$ and for $0 \leq j < k$, $\pi^j \models g_1$.

Standard Abbreviations

The customary abbreviations will be used for the connectives of propositional logic.

In addition, we will use the following abbreviations in writing temporal operators:

- $\mathbf{A}(f) \equiv \neg\,\mathbf{E}(\neg f)$
- ~~$f \equiv (true\ \mathbf{U}\ f)$~~ F f = (true U f)
- $\mathbf{G}\ f \equiv \neg\,\mathbf{F}\,\neg f$

CTL and LTL

CTL is a restricted subset of CTL* that permits only branching-time operators—each of the linear-time operators **G**, **F**, **X**, and **U** must be immediately preceded by a path quantifier.

Example: $\mathbf{AG}(\mathbf{EF}\, p)$

LTL consists of formulas that have the form $\mathbf{A}\, f$ where f is a path formula in which the only state subformulas permitted are atomic propositions.

Example: $\mathbf{A}(\mathbf{FG}\, p)$

Expressive Power

It can be shown that the three logics discussed in this section 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

There are eight basic CTL operators:

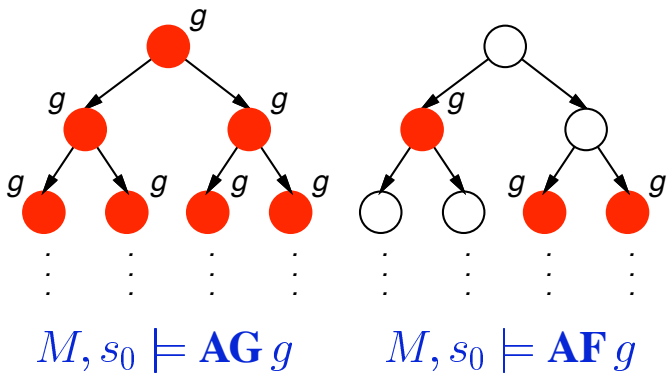
- **AX** and **EX**,
- **AG** and **EG**,
- **AF** and **EF**,
- **AU** and **EU**

Each of these can be expressed in terms of **EX**, **EG**, and **EU**:

- $\mathbf{AX}\, f = \neg\, \mathbf{EX}(\neg f)$
- $\mathbf{AG}\, f = \neg\, \mathbf{EF}(\neg f)$
- $\mathbf{AF}\, f = \neg\, \mathbf{EG}(\neg f)$
- $\mathbf{EF}\, f = \mathbf{E}[\mathit{true}\ \mathbf{U}\ f]$
- $\mathbf{A}[f\ \mathbf{U}\ g] \equiv \neg\, \mathbf{E}[\neg g\ \mathbf{U}\ \neg f \wedge \neg g] \wedge \neg\, \mathbf{EG}\, \neg g$

Basic CTL Operators

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



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

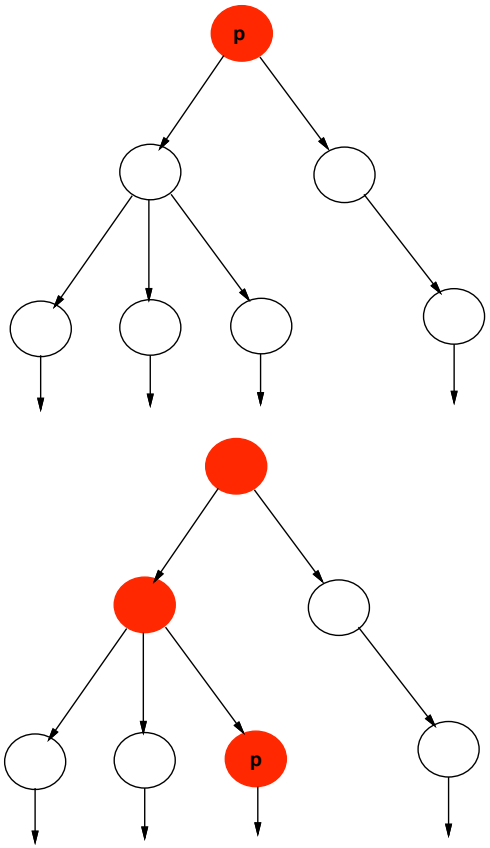
$M, s_0 \models \mathbf{EF}\, g$ $M, s_0 \models \mathbf{EG}\, g$

Typical CTL Formulas

- **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.

Fixpoint Algorithms

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



Fixpoint Algorithms (cont.)

Key properties of $\mathbf{EF}\, p$:

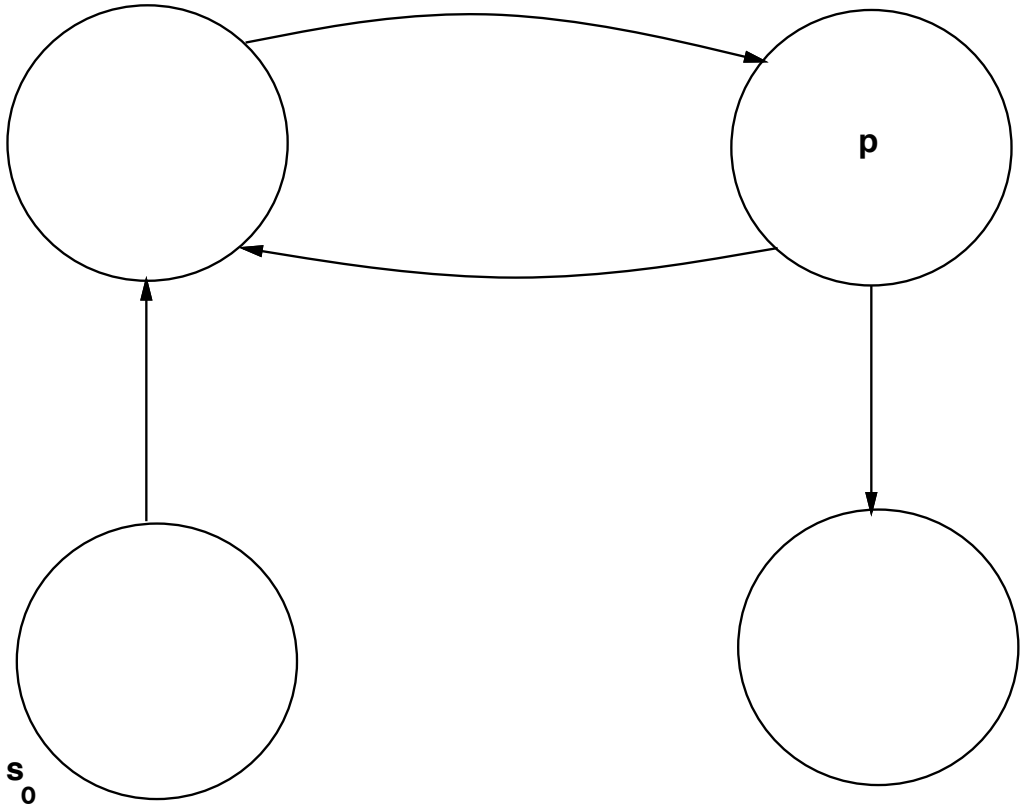
1. $\mathbf{EF}\, p = p \vee \mathbf{EX}\, \mathbf{EF}\, p$
2. $U = p \vee \mathbf{EX}\, U$ implies $\mathbf{EF}\, p \subseteq U$

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

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

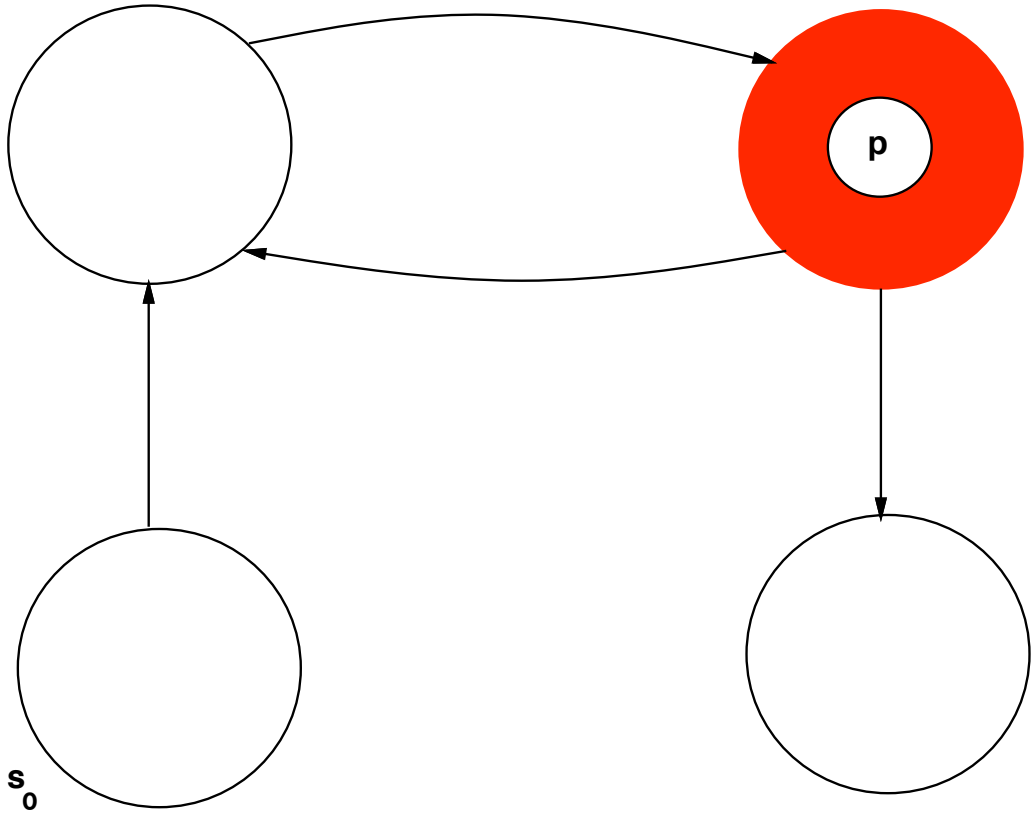
$$
\begin{aligned}
U_0 &= \mathbf{False} \\
U_1 &= p \vee \mathbf{EX}\, U_0 \\
U_2 &= p \vee \mathbf{EX}\, U_1 \\
U_3 &= p \vee \mathbf{EX}\, U_2 \\
&\vdots
\end{aligned}
$$

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



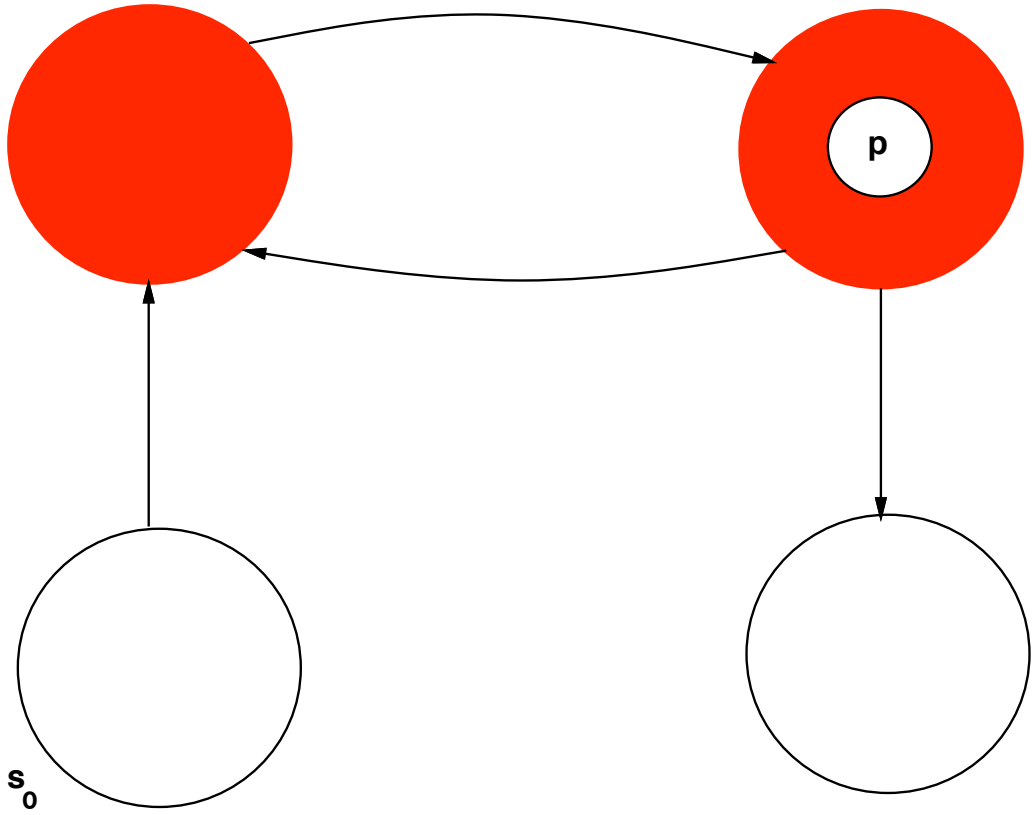
$U_0 = \emptyset$

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



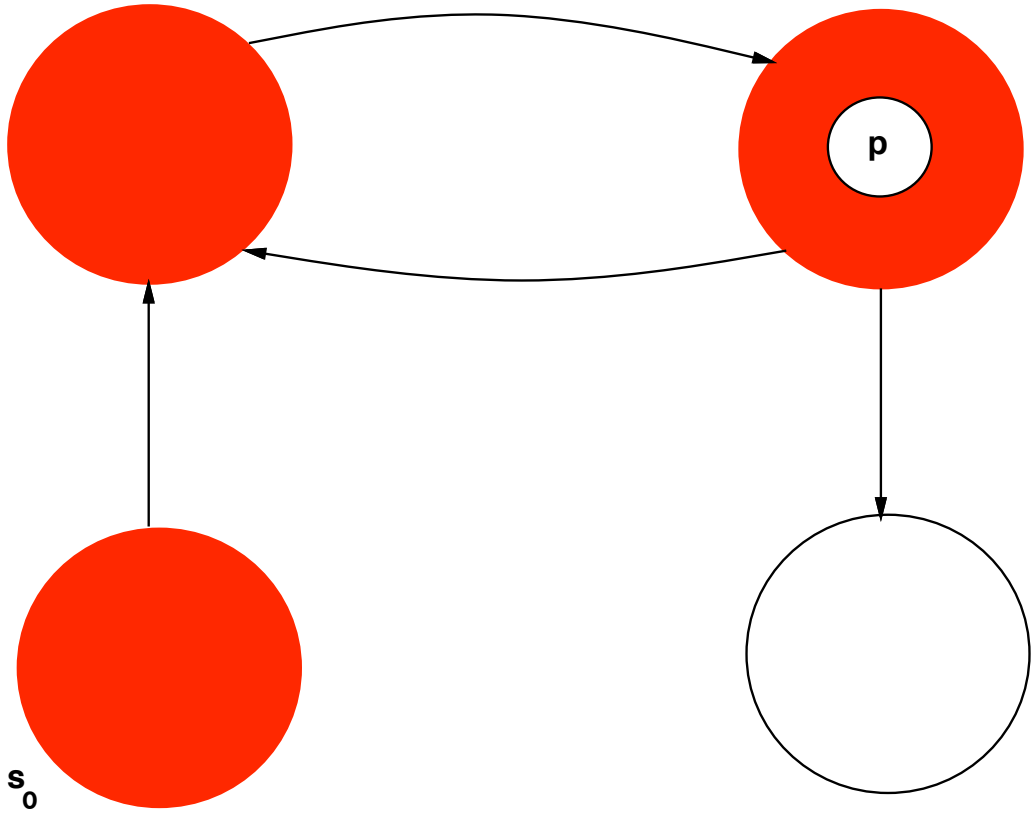
$$U_1 = p \vee \mathbf{EX}\, U_0$$

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



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

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



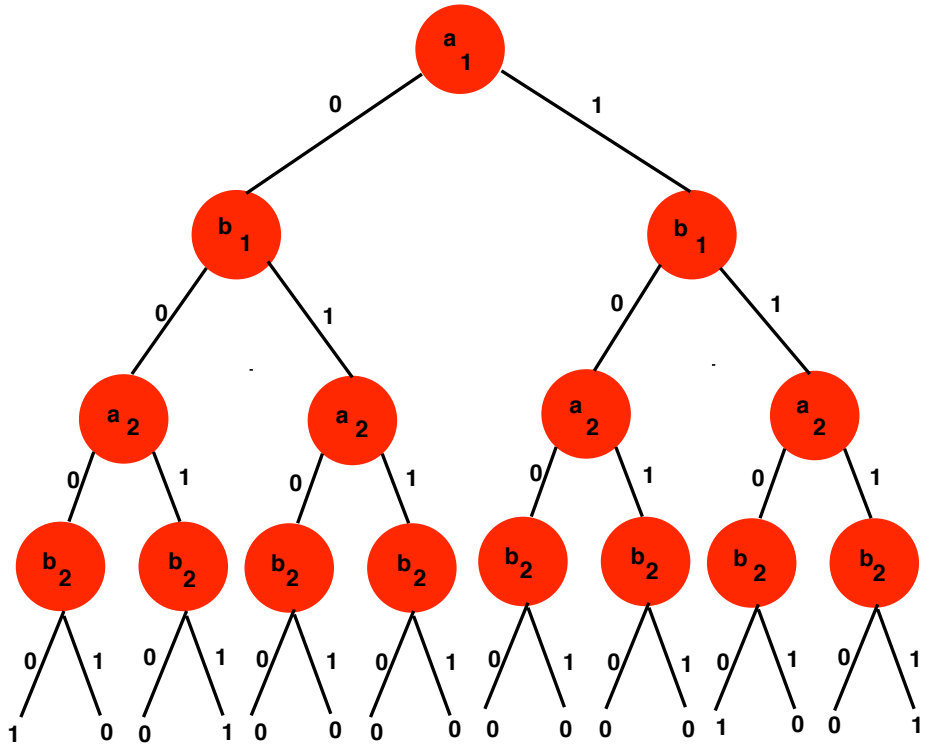
$U_3 = p \vee \mathbf{EX}\, U_2$

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

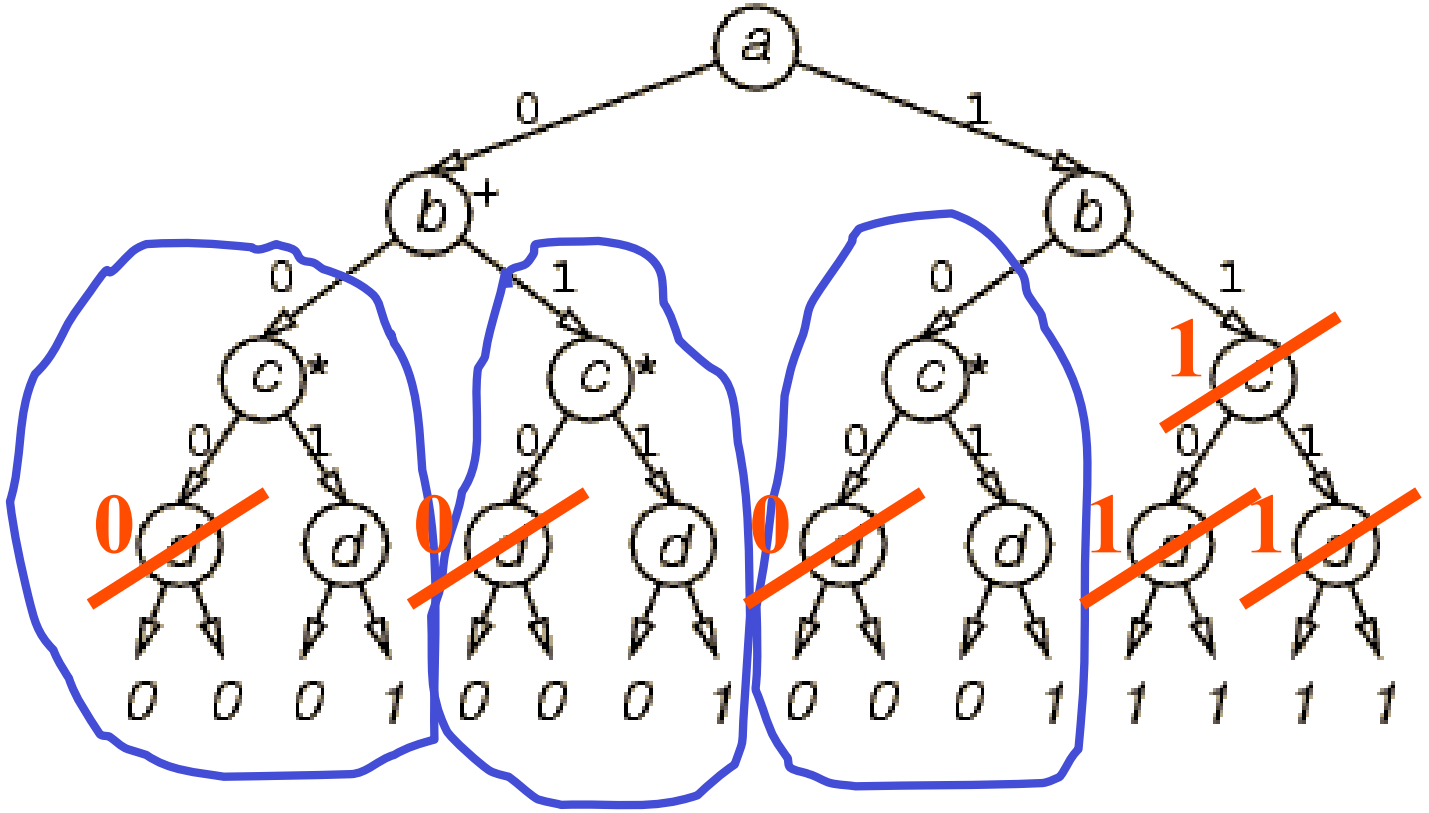
An Ordered Binary Decision Diagram (OBDD) is an ordered decision tree where

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

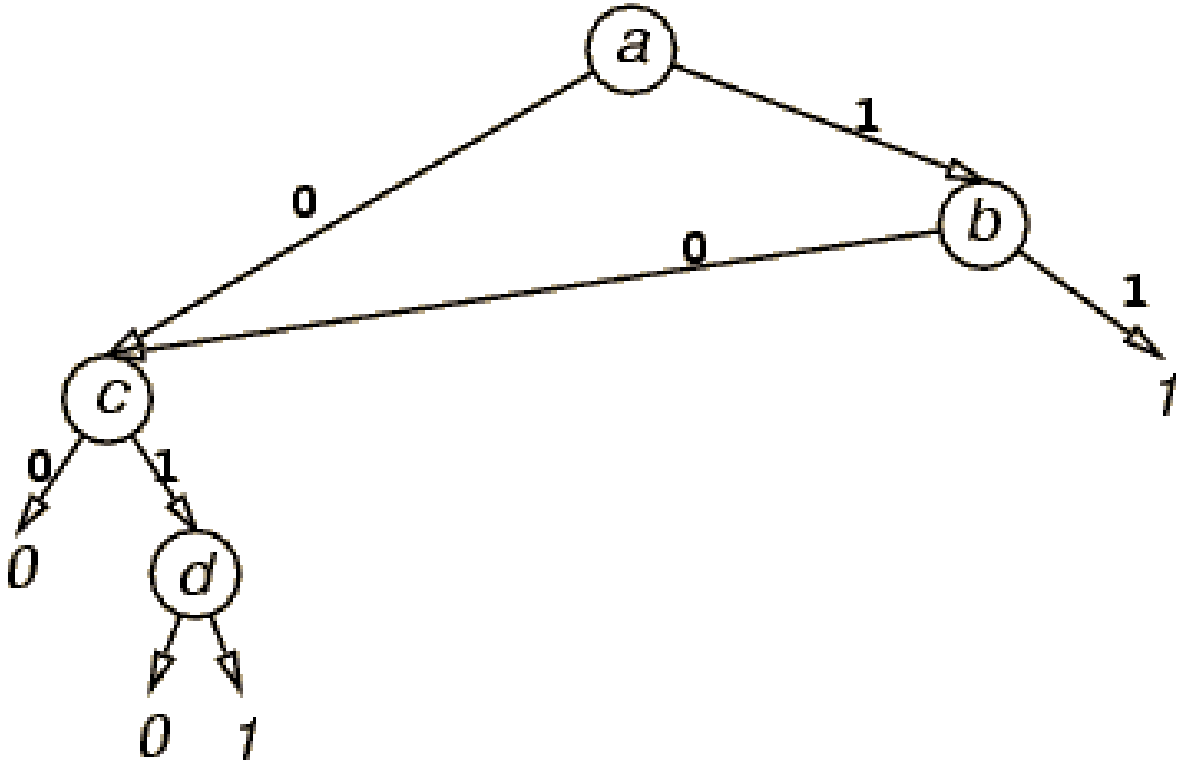
Given a parameter ordering, OBDD is unique up to isomorphism.

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

Ordered decision tree for the function $ab + cd$:

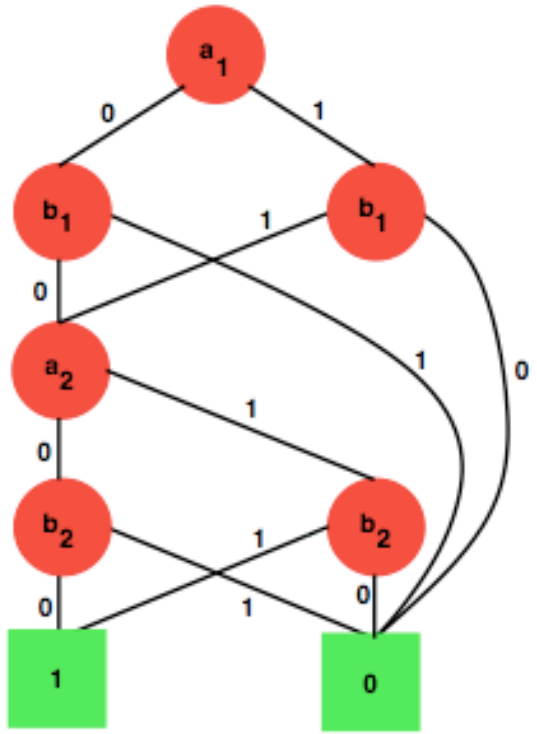


Binary Decision Diagram - BDD



OBDD for Comparator Example

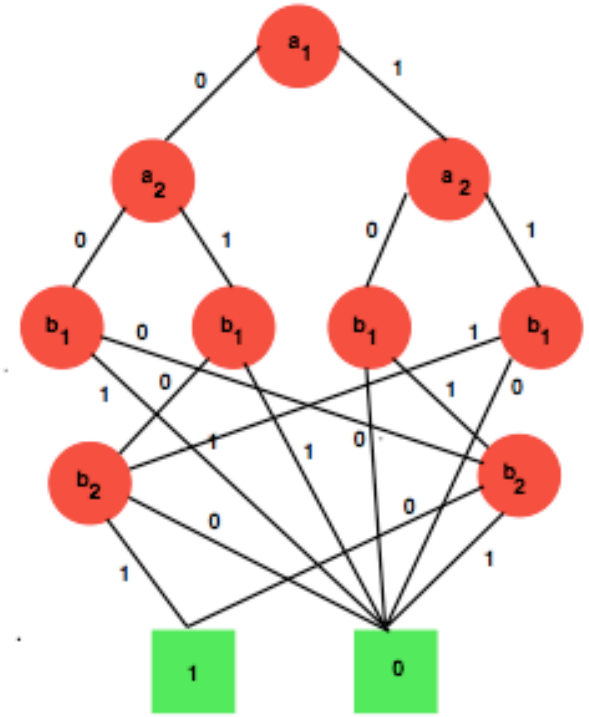
If we use the ordering $a_1 < b_1 < a_2 < b_2$ for the comparator function, we obtain the OBDD below:



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:

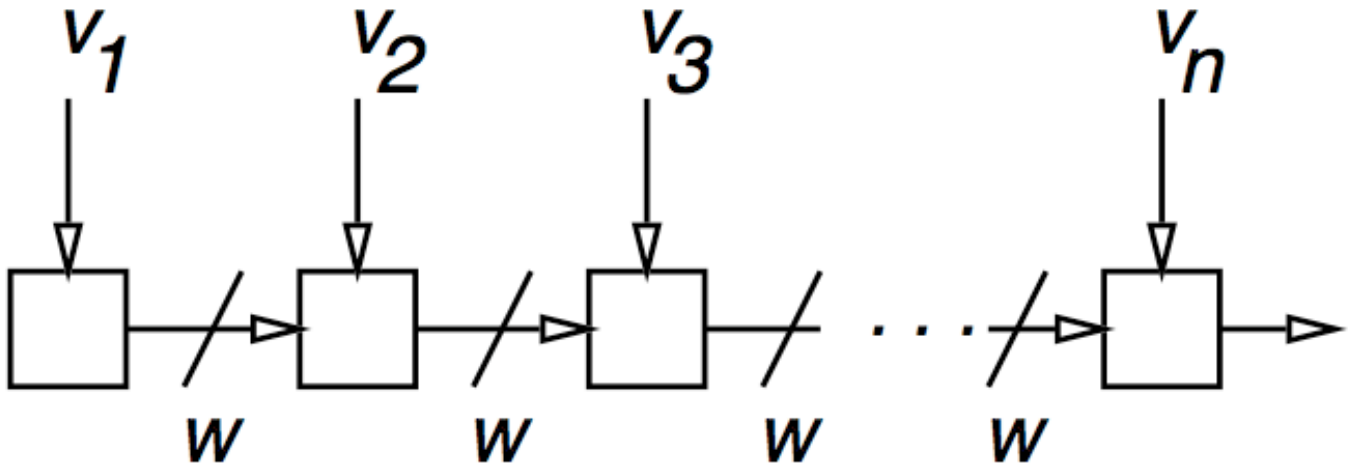
- if we use the ordering $a_1 < b_1 < \ldots < a_n < b_n$, the number of vertices will be $3n + 2$.
- if we use the ordering $a_1 < \ldots < a_n < b_1 \ldots < b_n$, the number of vertices is $3 \cdot 2^n - 1$.

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.

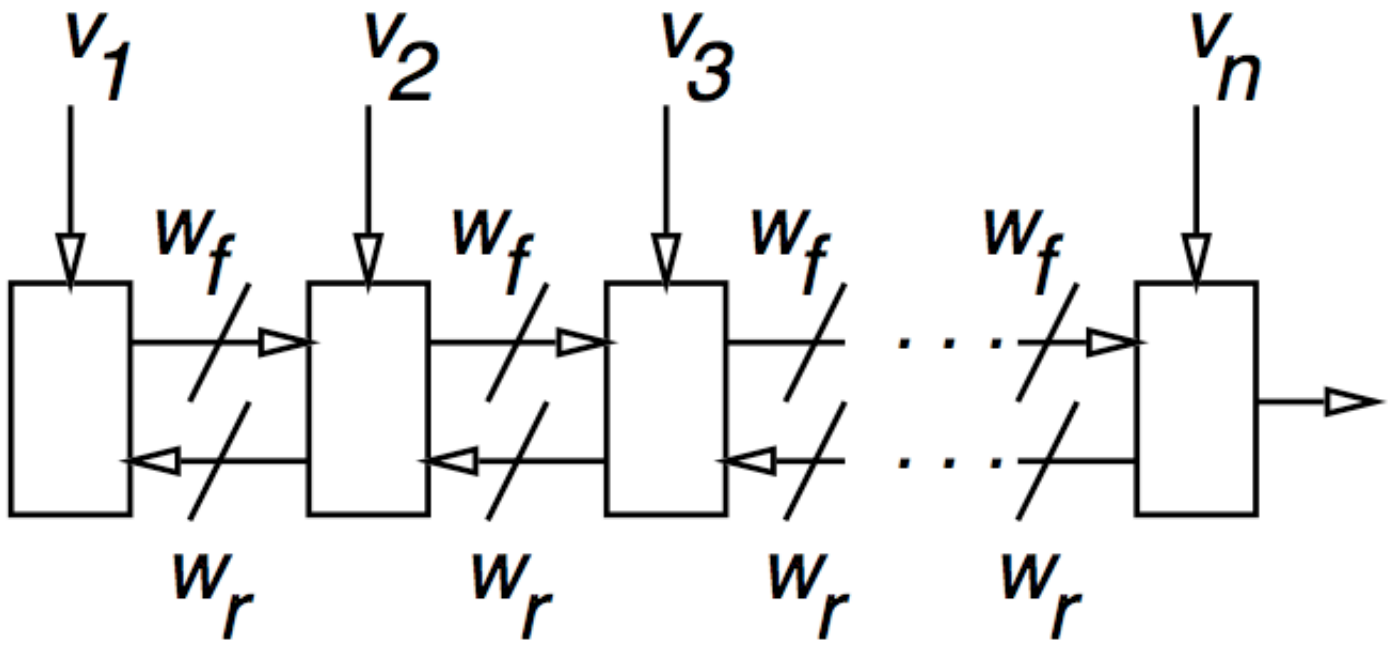
2.3 Circuit width and OBDD size

For a given topological sort of of the gates of a circuit, the *width* is the maximum number of wires crossing any cut:



For the corresponding order of the input variables, the OBDD size is bounded from above by $n2^w$. That is, as we increase circuit size, keeping the width constant, the OBDD grows at most linearly.

This is easy to see: having traversed the first i variables in the OBDD order, the function of the remaining variables is determined by the value of at most w wires. Since there are only 2^w values of w wires, there are at most 2^w inequivalent nodes at level i of the OBDD.



we have OBDD size $\leq n2^{w_f 2^{w_r}}$ (still linear in n).

A good example of a bounded width circuit is an adder, where only a single wire (the carry bit) passes from one adder cell to the next. Thus, the OBDD representation for any output bit of an adder is $O(n)$. The same argument applies to a parity chain.

A multiplier, on the other hand, has no constant width impementation, and consequently no efficient OBDD representation for any variable order.

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,

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

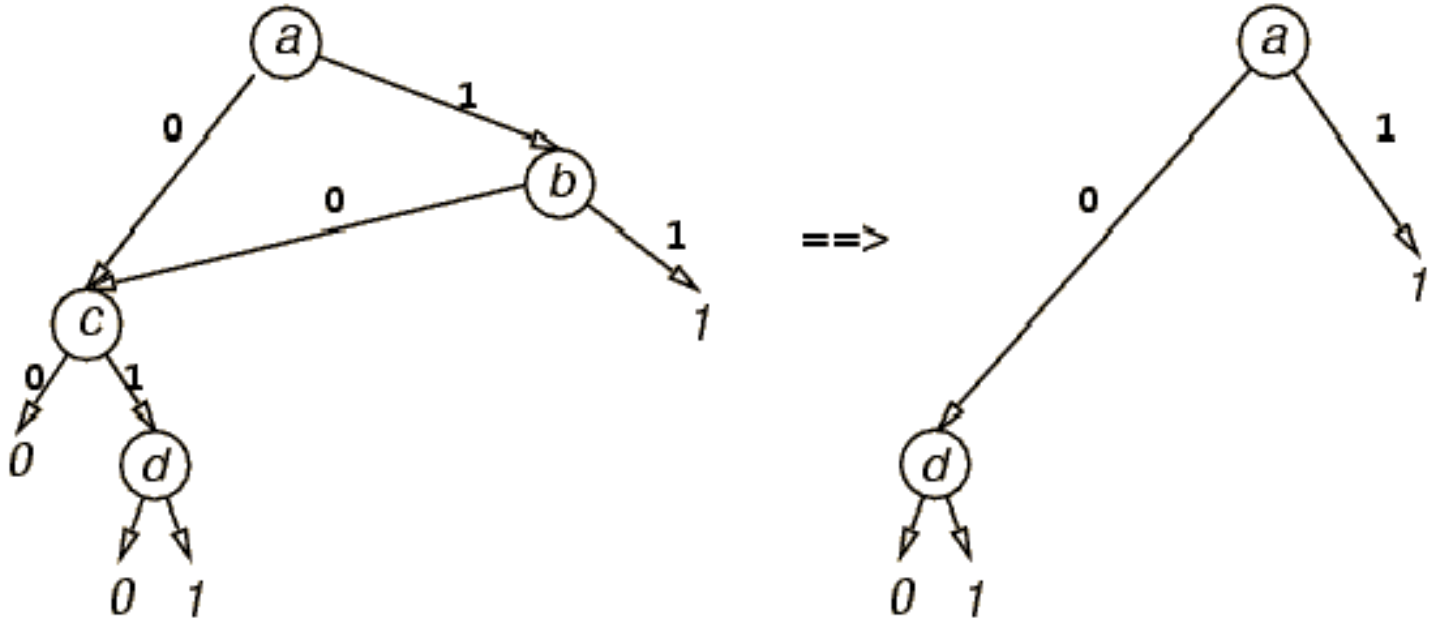
  - $f(a, b, c, d)|_{\bar{a}}$: replace all a nodes by left sub-tree.
  - $f(a, b, c, d)|_{a}$: replace all a nodes by right sub-tree.

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

$$\exists(w_1, w_2, \ldots, w_n).\ f$$

This is done by recursively replacing each OBDD node labeled with w_i with the logical or of its two branches.

Example: $\exists(b, c).\ (ab \vee cd) \ = \ c \vee d$



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:

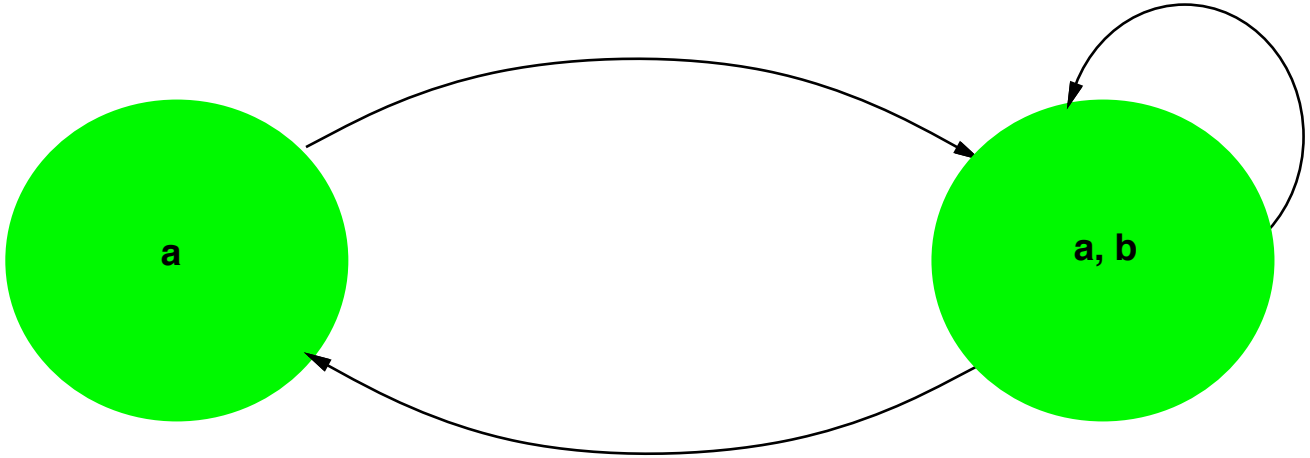
$$T(v_1, \ldots, v_n, v'_1, \ldots, v'_n)$$

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:

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

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.