Succinct Representations (BDDs and SAT)

CS60030 FORMAL SYSTEMS

PALLAB DASGUPTA,
FNAE, FASc,
A K Singh Distinguished Professor in AI,
Dept of Computer Science & Engineering
Indian Institute of Technology Kharagpur
Email: pallab@cse.iitkgp.ac.in
Web: http://cse.iitkgp.ac.in/~pallab

FORMAL METHODS FOR SAFETY CRITICAL SYSTEMS

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Set Membership versus Boolean Functions

- Suppose state variables are x_1, x_2, x_3 and states are encoded as $\langle x_1\ x_2\ x_3 \rangle$
- Consider the set of states: S = { 000, 010, 011, 100, 101 }
- Boolean membership function for S: $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1 x_2 \bar{x}_3 + \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{x}_2 x_3$

- Why use Boolean functions to represent state sets?
  - Because Boolean functions can be minimized
  - Often size of a circuit is logarithmic in the number of minterms
- $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1 x_2 \bar{x}_3 + \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{x}_2 x_3 = \bar{x}_1\bar{x}_3 + \bar{x}_1 x_2 + x_1 \bar{x}_2$

Representations of Boolean Functions

- Disjunctive Normal Form (Sum of minterms)

$$f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_3 + \bar{x}_1 x_2 + x_1\bar{x}_2$$

  - Checking satisfiability is easy, checking validity is hard

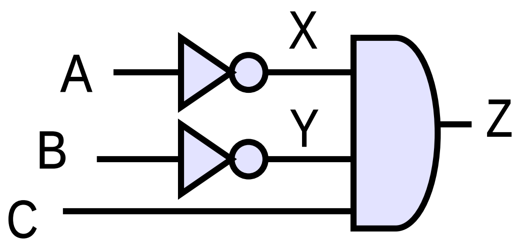
- Conjunctive Normal Form (Product of clauses)

$$g(x_1, x_2, x_3) = (\bar{x}_1 + \bar{x}_3)(\bar{x}_1 + x_2)(x_1 + \bar{x}_2)$$

  - Checking validity is easy, checking satisfiability is har

- Translation between CNF and DNF is computationally hard

Converting a Circuit to SAT



A circuit describes the relationship (constraints) between its nets

p=q can be written as $(\boldsymbol{p} + \overline{\boldsymbol{q}})(\overline{\boldsymbol{p}} + \boldsymbol{q})$

CLAUSE FORM:

The circuit functionality is: $(\boldsymbol{x} = \overline{\boldsymbol{a}})(\boldsymbol{y} = \overline{\boldsymbol{b}})(\boldsymbol{z} = \boldsymbol{xyc})$

which may be rewritten as:

$$(\boldsymbol{x} + \boldsymbol{a})(\overline{\boldsymbol{x}} + \overline{\boldsymbol{a}})(\boldsymbol{y} + \boldsymbol{b})(\overline{\boldsymbol{y}} + \overline{\boldsymbol{b}})(\boldsymbol{z} + \overline{\boldsymbol{x}} + \overline{\boldsymbol{y}} + \overline{\boldsymbol{c}})(\overline{\boldsymbol{z}} + \boldsymbol{x})(\overline{\boldsymbol{z}} + \boldsymbol{y})(\overline{\boldsymbol{z}} + \boldsymbol{c})$$

Typically the number of clauses for a circuit is much smaller than 2^n (the number of rows in the truth table).

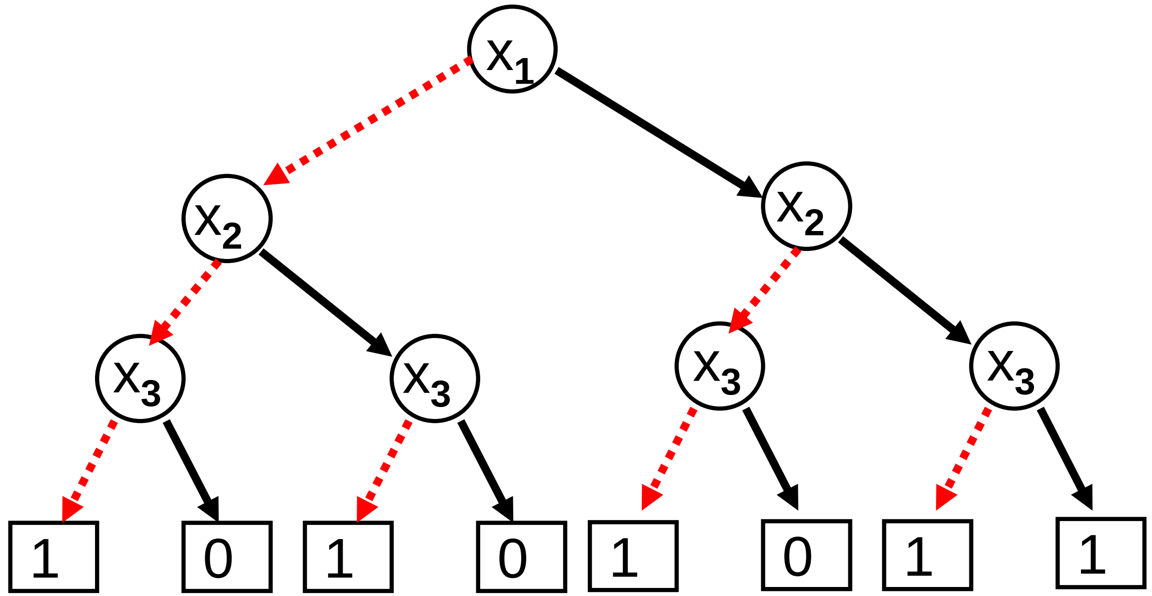
Binary Decision Diagrams (BDDs)

Graphical representation [Lee, Akers, Bryant]

- Efficient representation & manipulation of Boolean functions in many practical cases
- Enables efficient verification/analysis of a large class of designs
- Worst-case behavior still exponential

Example: $f = (x_1 \wedge x_2) \vee \neg x_3$

- Represent as binary tree
- Evaluating f:
    - Start from root
    - For each vertex labeled x_i
        - take dotted branch if $x_i = 0$
        - else take solid branch

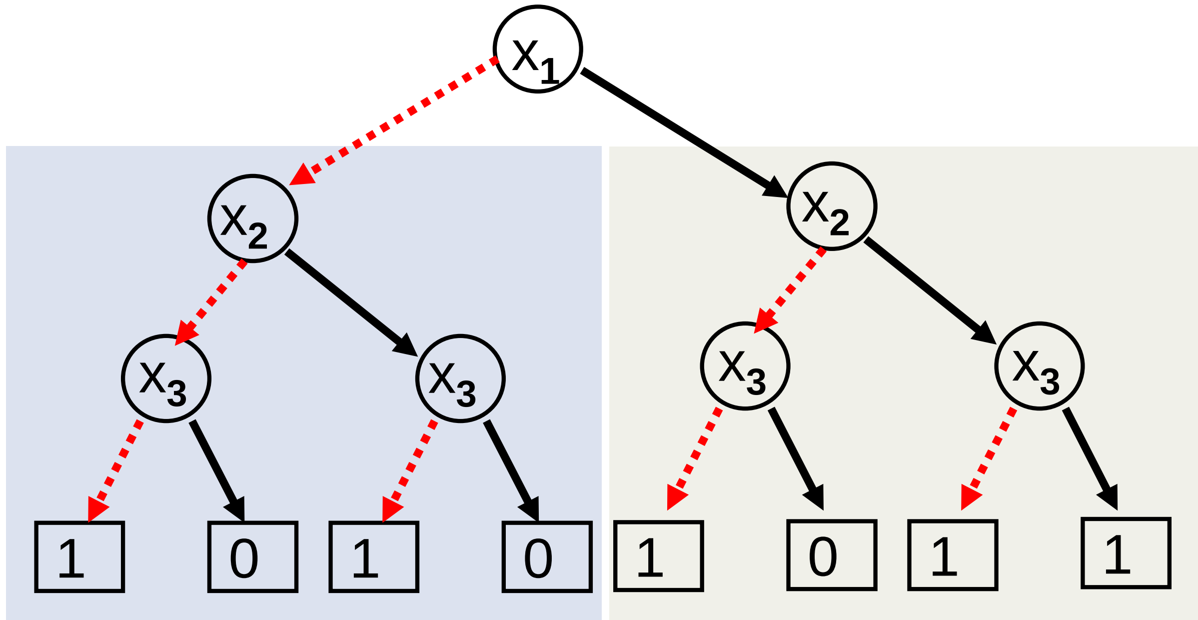


Binary Decision Diagrams (BDDs)

Underlying principle: Shannon decomposition

- $f(x_1, x_2, x_3) = x_1 \wedge f(1, x_2, x_3) \vee \neg x_1 \wedge f(0, x_2, x_3)$
  $= x_1 \wedge (x_2 \vee \neg x_3) \vee \neg x_1 \wedge (\neg x3)$
- Can be applied recursively to $f(1, x_2, x_3)$ and $f(0, x_2, x_3)$
  - Gives tree
- Extend to n arguments

Number of nodes can be exponential in number of variables



$f = (x_1 \wedge x_2) \vee \neg x_3$

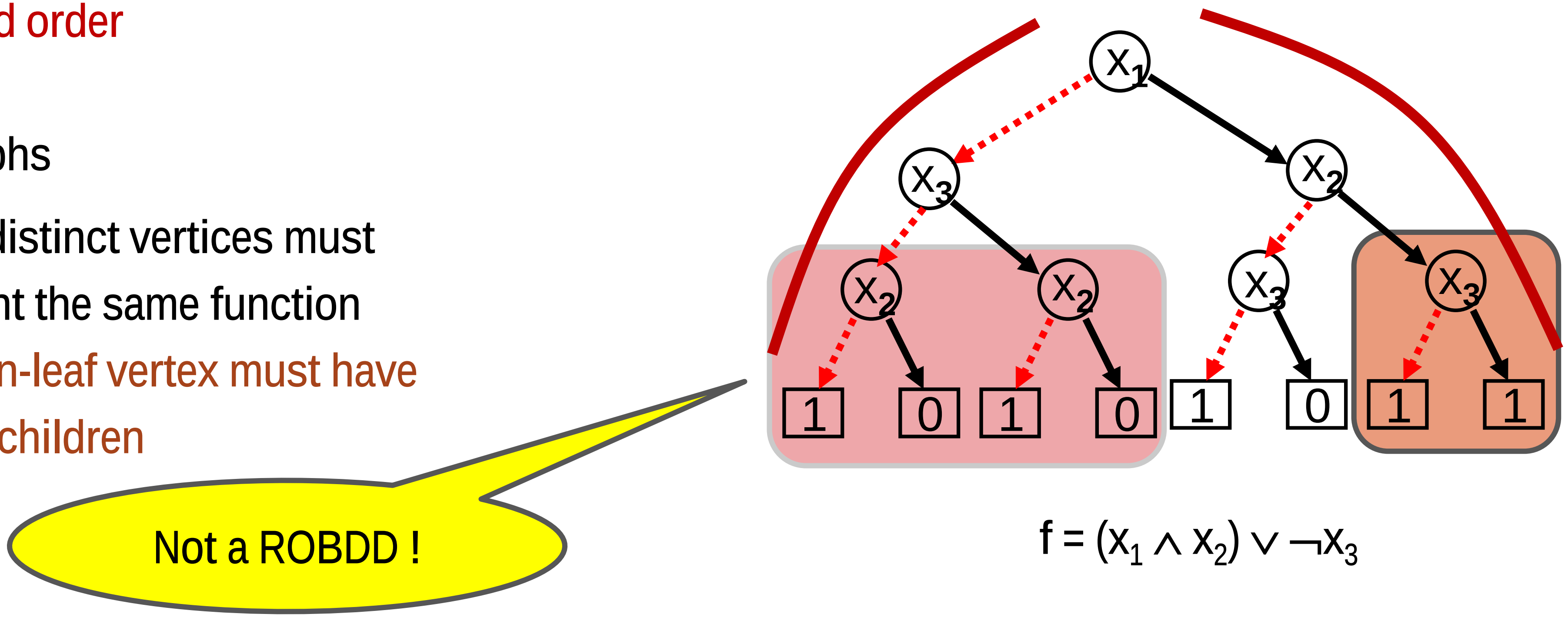
Restrictions on BDDs

Ordering of variables

- In all paths from root to leaf, variable labels of nodes must appear in a specified order

Reduced graphs

- No two distinct vertices must represent the same function
- Each non-leaf vertex must have distinct children

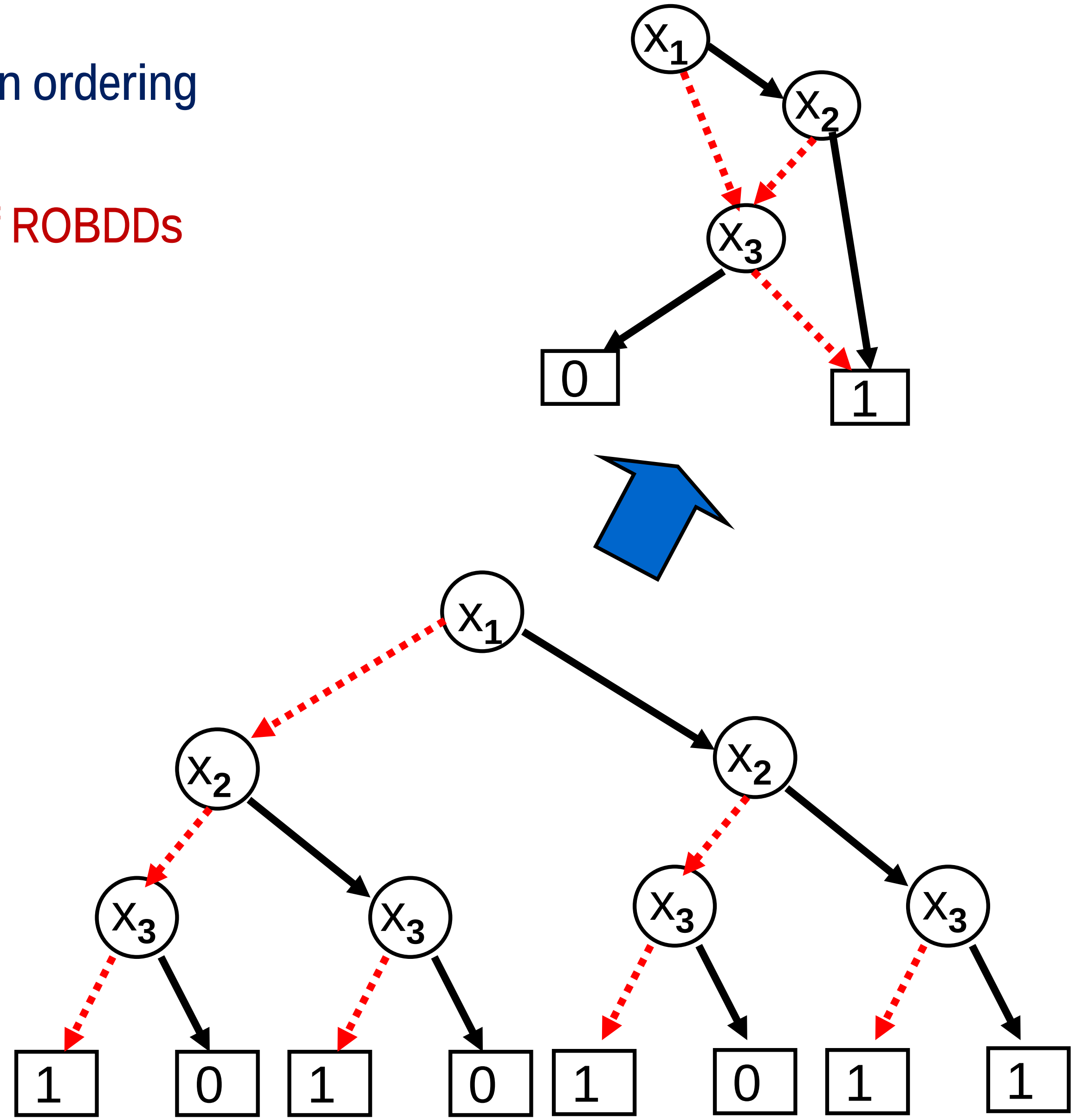


$f = (x_1 \wedge x_2) \vee \neg x_3$

REDUCED ORDERED BDD (ROBDD): Directed Acyclic Graph

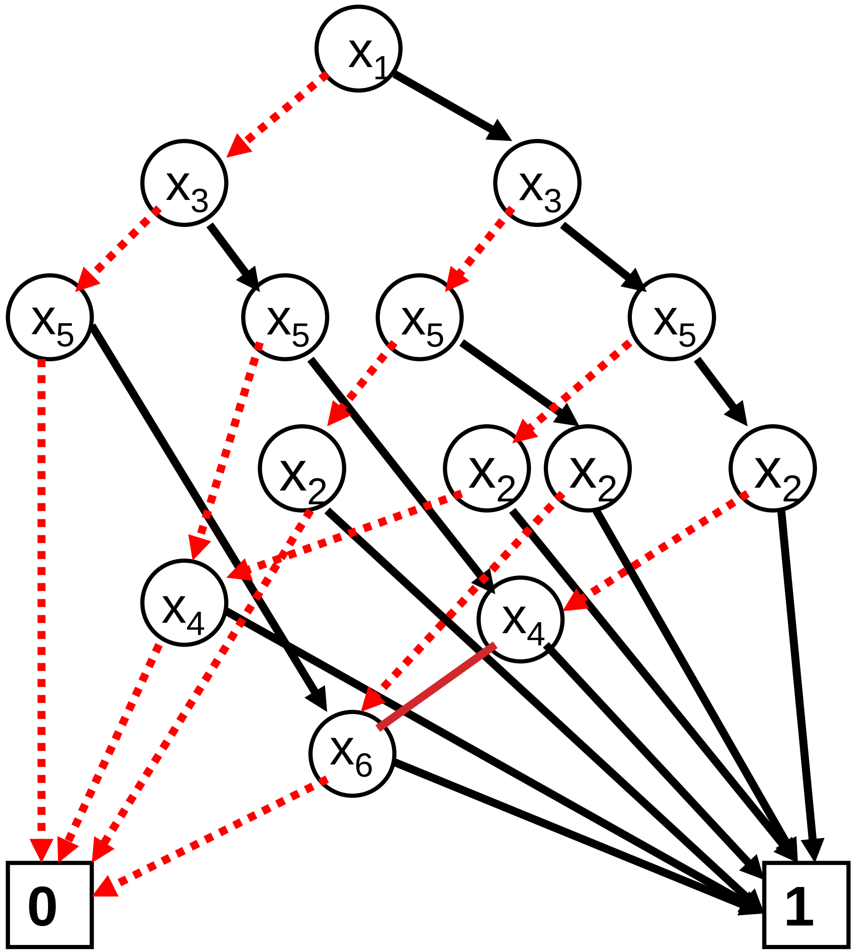
ROBDDs

- Unique (canonical) representation of f for given ordering of variables
    - Checking f1 = f2 reduces to checking if ROBDDs are isomorphic
- Shared subgraphs: size reduction
- Every path doesn't have all labels x1, x2, x3
- Every non-leaf vertex has a path to 0 and 1

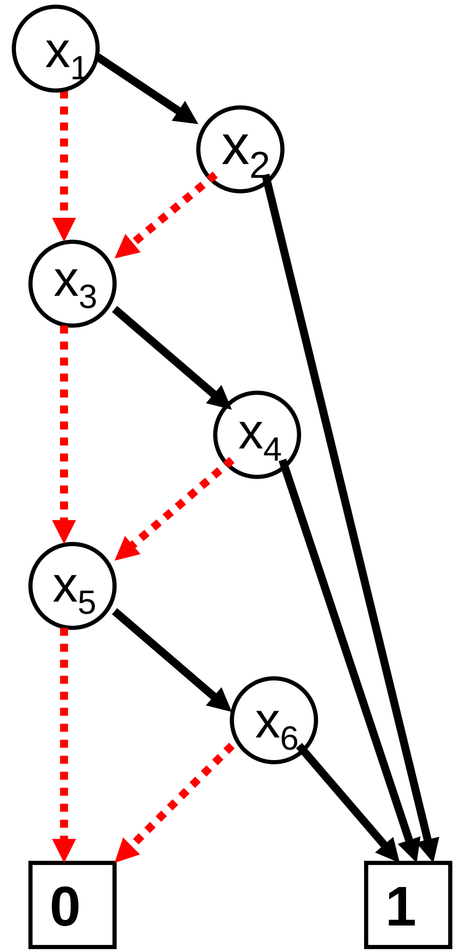


Variable Ordering Problem

$$f = x_1x_2 + x_3x_4 + x_5x_6$$



Order: $x_1 < x_3 < x_5 < x_2 < x_4 < x_6$



Order: $x_1 < x_2 < x_3 < x_4 < x_5 < x_6$

Variable Ordering Problem

ROBDD size

- Extremely sensitive to variable ordering
  - $f = x_1x_2 + x_3x_4 + \ldots + x_{2n-1}x_{2n}$
    - 2n+2 vertices for order $x_1 < x_2 < x_3 < x_4 < \ldots x_{2n-1} < x_{2n}$
    - 2^{n+1} vertices for order $x_1 < x_{n+1} < x_2 < x_{n+2} < \ldots x_n < x_{2n}$
  - $f = x_1\, x_2\, x_3 \ldots .\, x_n$
    - n+2 vertices for all orderings
  - Exponential regardless of variable ordering
    - Most significant bit of product of n-bit integer multiplier [Bryant]

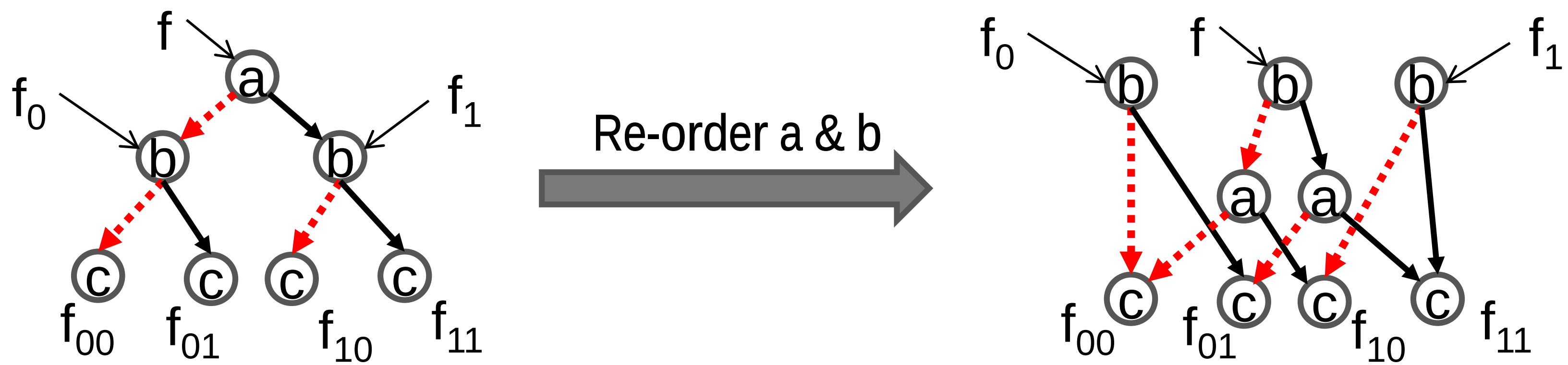
Determining best variable order for arbitrary functions is computationally intractable

- Heuristics: Static ordering, Dynamic ordering

Variable Ordering Solutions

Dynamic ordering

- Starts with user-provided static order
- If dynamic re-ordering triggered on-the-fly, evaluate benefits of re-ordering small subset of variables
    - If beneficial, re-order and repeat until no benefit
- Expensive in general, sophisticated triggers essential
- Key observation [Friedman]: Given ROBDD with $x_1 < \ldots x_i < x_{i+1} < \ldots x_n$,
    - Permuting $x_1 \ldots x_i$ has no effect on ROBDD nodes labeled by $x_{i+1} \ldots x_n$
    - Permuting $x_{i+1} \ldots x_n$ has no effect on ROBDD nodes labeled by $x_1 \ldots x_i$
    - Variables in adjacent levels easily swappable



How to use a BDD package

$$f(x,a,b,c,z) = (x+a)(\overline{x}+\overline{a})(y+b)(\overline{y}+\overline{b})(z+\overline{x}+\overline{y}+\overline{c})(\overline{z}+x)(\overline{z}+y)(\overline{z}+c)$$

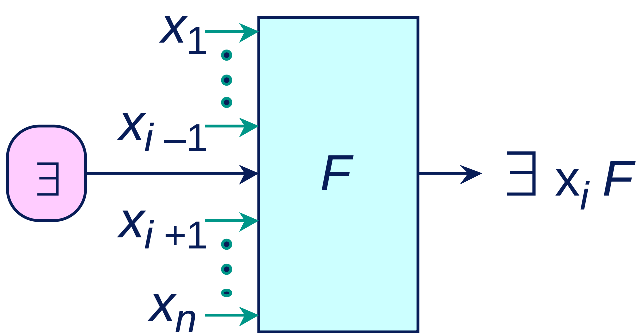
- Create a BDD manager
- Create BDDs of sub-functions and then the functions

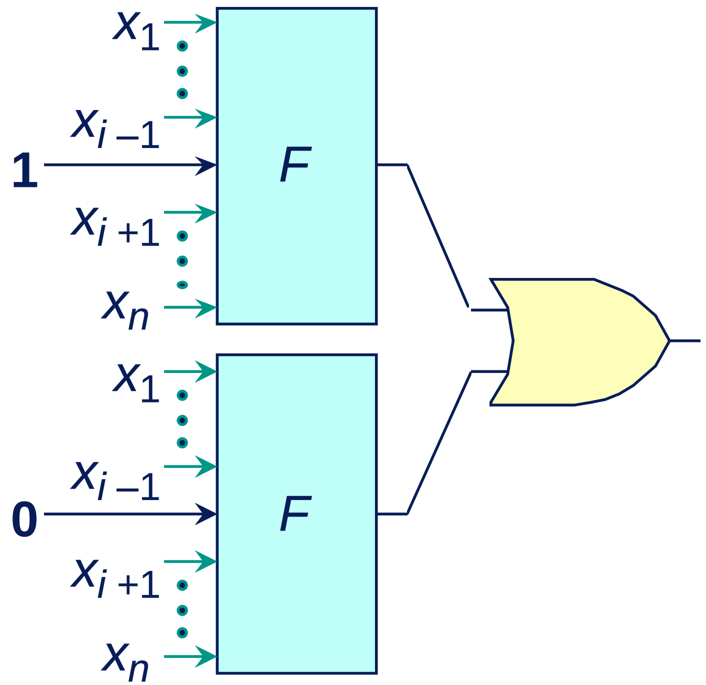
```
bdd1 = Cudd_bddOr(gbm, x, a);
bdd2 = Cudd_bddOr(gbm, y, b);
bdd3 = Cudd_bddAnd(gbm, bdd1, bdd2);
... and so on.
```

- More to be discussed during hands-on sessions

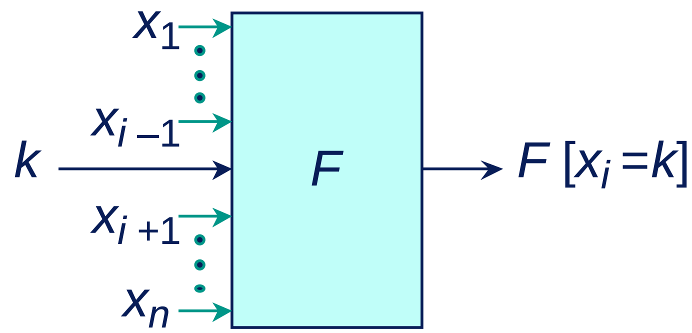
BDD Operations

- All logical operations – *AND, OR, NOT, etc.*
- Validity Checking: The BDD of a valid function reduces to the single node 1
- Satisfiability Checking: The BDD of an unsatisfiable function reduces to the single node 0
- Variable Quantification:

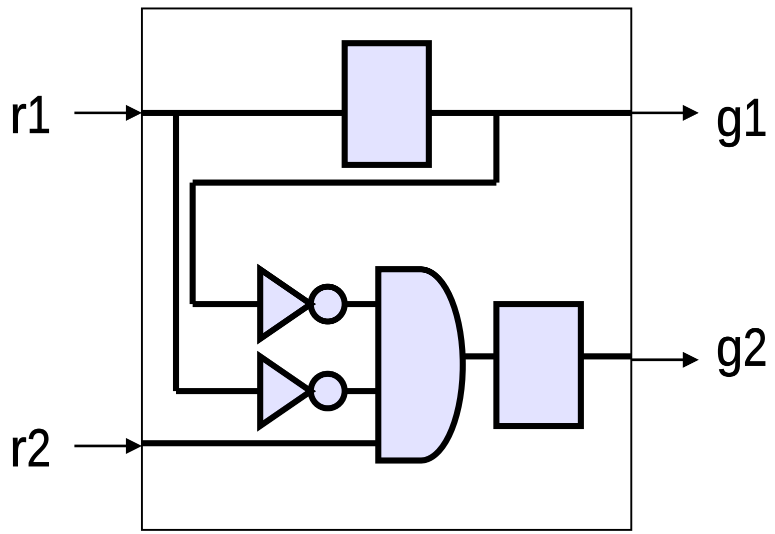


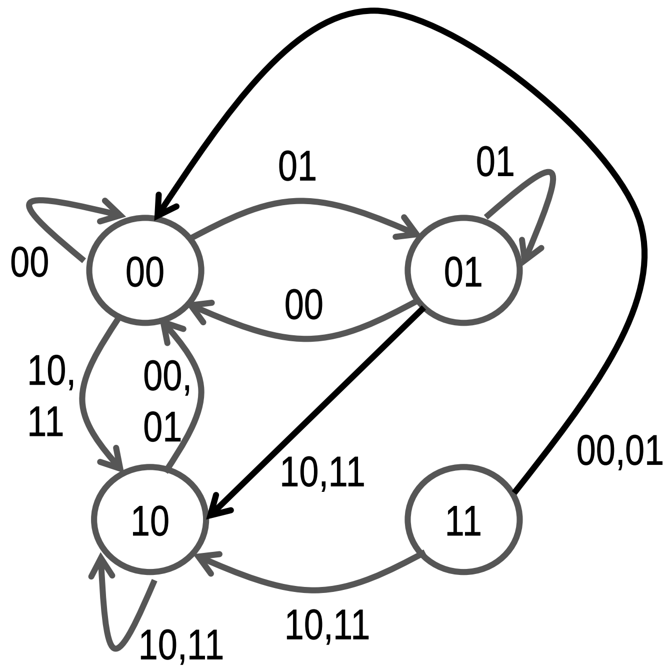


- Restrict operation: *Effect of setting function argument x_i to constant k (0 or 1).*
  - Also called Cofactor operation



Basics of Finite State Systems





PS g_1g_2	I/P r_1r_2	NS $g'_1g'_2$
00	00	00
00	01	01
00	10	10
00	11	10
01	00	00
01	01	01
01	10	10
01	11	10
10	00	00
10	01	00
10	10	10
10	11	10
11	00	00
11	01	00
11	10	10
11	11	10

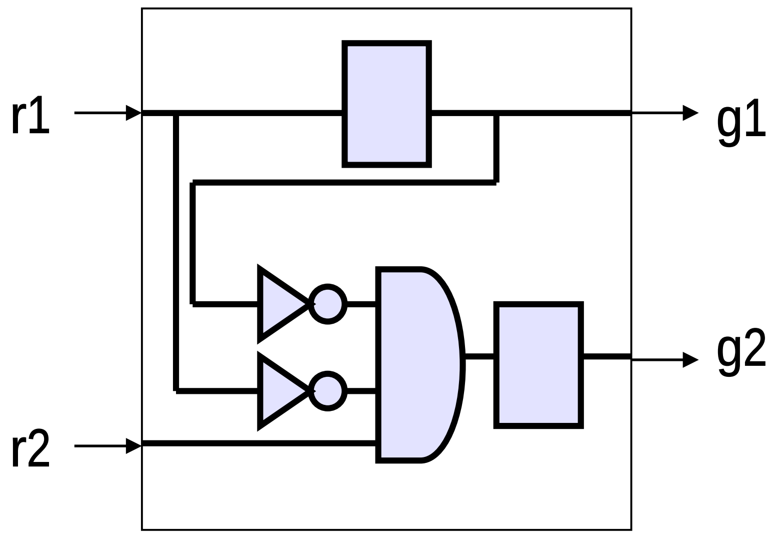
Transition Relation:

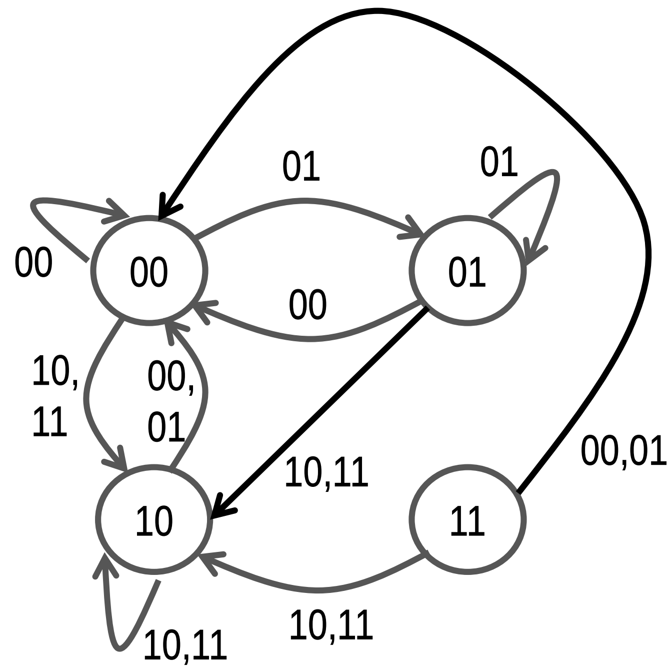
$g'_1 \Leftrightarrow r_1$

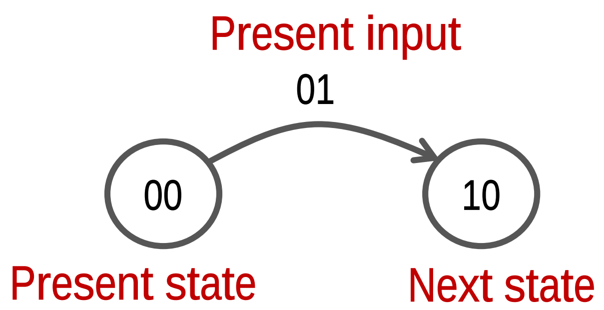
$g'_2 \Leftrightarrow \neg r_1 \wedge r_2 \wedge \neg g_1$

Initial State: $r_1=0, r_2=0, g_1=0, g_2=1$

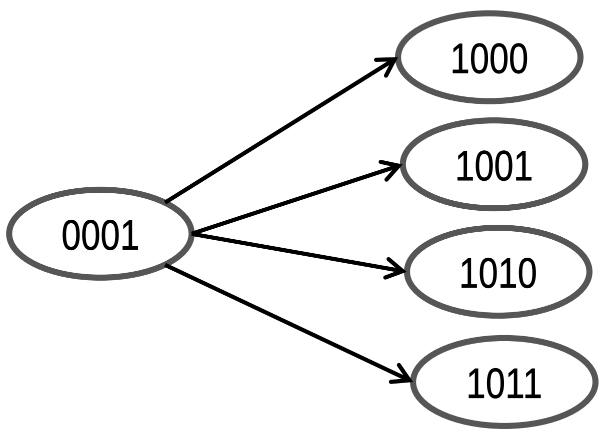
Open Systems versus Non-Deterministic Closed Systems







The next input is non-deterministic



The input is part of the state. Since the next input is not known we have a non-deterministic state machine.

PS g_1g_2	I/P r_1r_2	NS $g'_1g'_2$	Next I/P
00	00	00	xx
00	01	01	xx
00	10	10	xx
00	11	10	xx
01	00	00	xx
01	01	01	xx
01	10	10	xx
01	11	10	xx
10	00	00	xx
10	01	00	xx
10	10	10	xx
10	11	10	xx
11	00	00	xx
11	01	00	xx
11	10	10	xx
11	11	10	xx

The complete *transition relation*

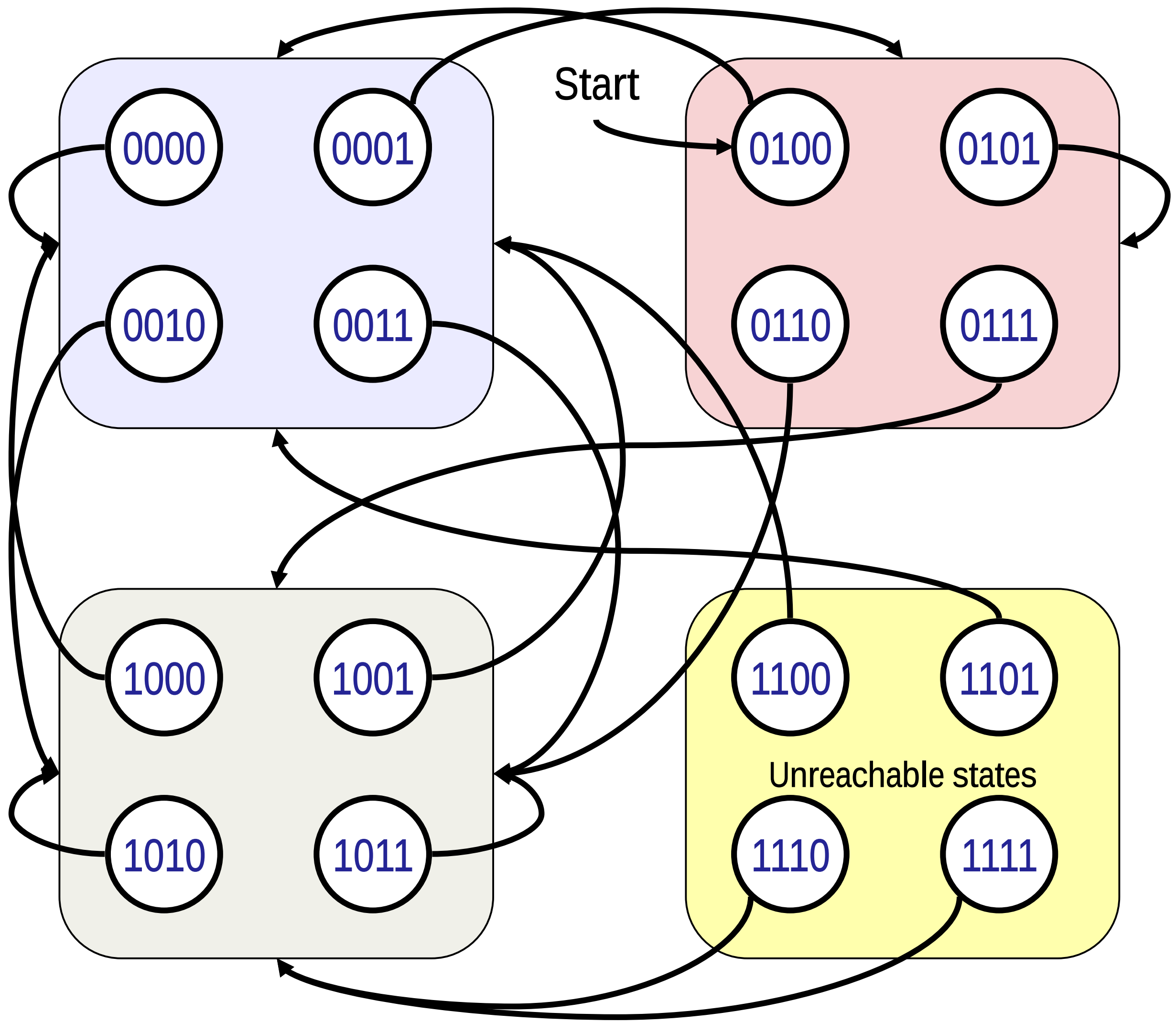
<u>*Transition Relation:*</u>

$g'_1 \Leftrightarrow r_1$

$g'_2 \Leftrightarrow \neg r_1 \wedge r_2 \wedge \neg g_1$

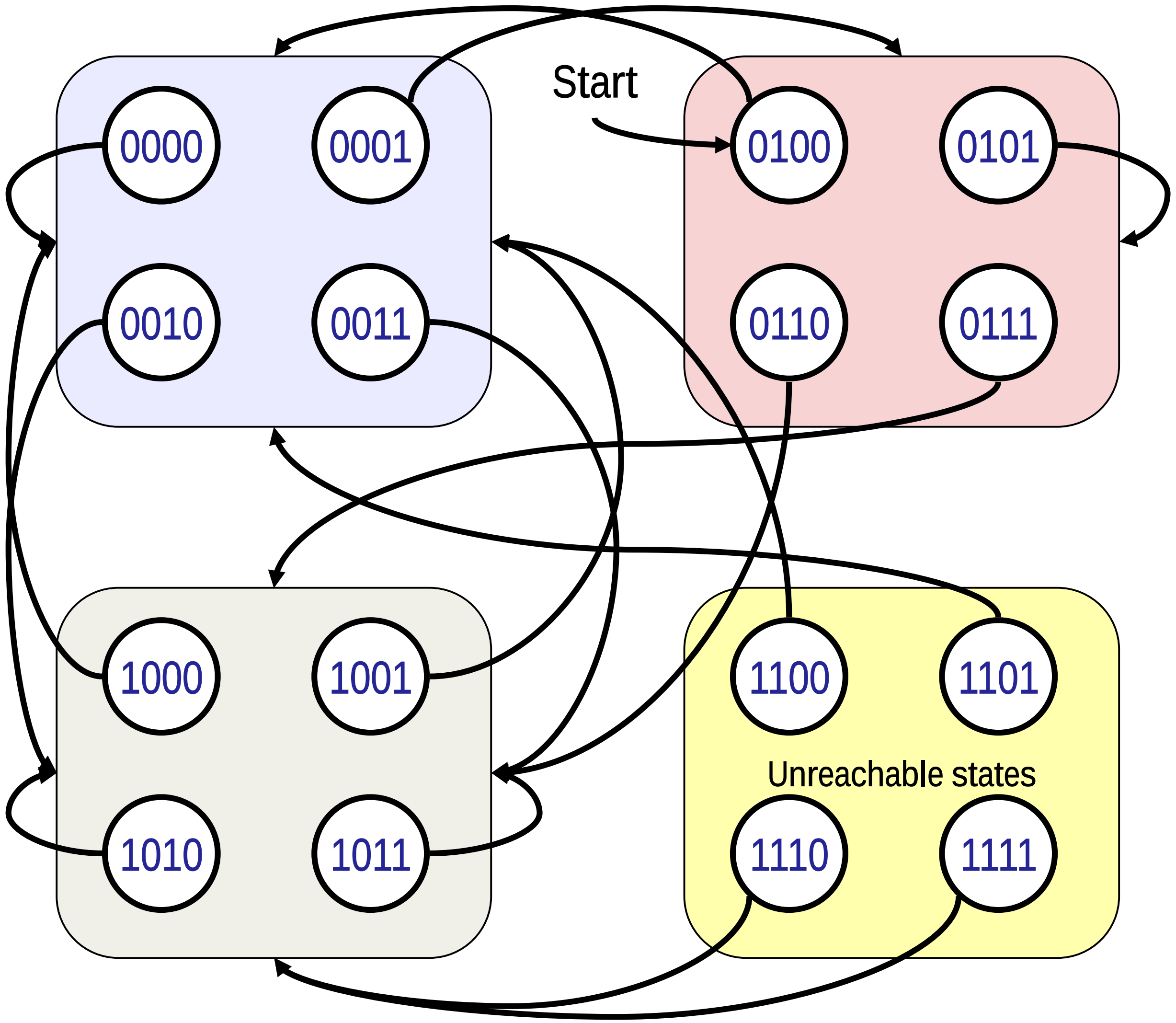
Initial State:

$r_1=0,\ r_2=0,\ g_1=0,\ g_2=1$



PS g_1g_2	I/P r_1r_2	NS $g'_1g'_2$	Next I/P
00	00	00	xx
00	01	01	xx
00	10	10	xx
00	11	10	xx
01	00	00	xx
01	01	01	xx
01	10	10	xx
01	11	10	xx
10	00	00	xx
10	01	00	xx
10	10	10	xx
10	11	10	xx
11	00	00	xx
11	01	00	xx
11	10	10	xx
11	11	10	xx

State Labels: Propositions



PS g_1g_2	I/P r_1r_2	NS $g'_1g'_2$	Next I/P
00	00	00	xx
00	01	01	xx
00	10	10	xx
00	11	10	xx
01	00	00	xx
01	01	01	xx
01	10	10	xx
01	11	10	xx
10	00	00	xx
10	01	00	xx
10	10	10	xx
10	11	10	xx
11	00	00	xx
11	01	00	xx
11	10	10	xx
11	11	10	xx

p: $g_1 \wedge g_2$

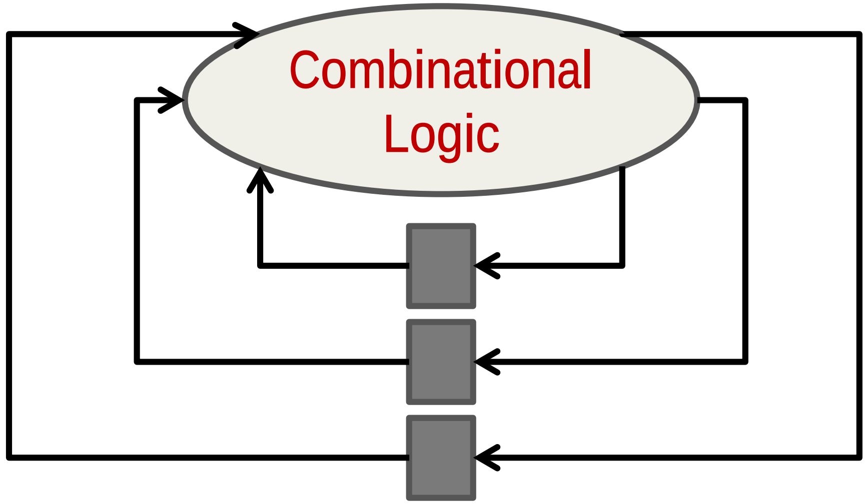
The states in the yellow box are labeled with p

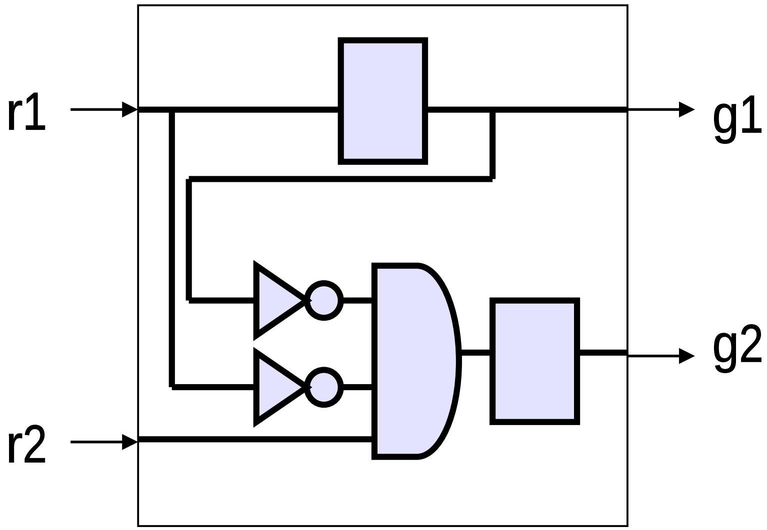
q: $r_1 = g_1$

The states labeled with q are 0000, 0001, 0100, 0101, 1010, 1011, 1110, 1111

Succinct representation of State Machines

- Sequential functions: Combinational logic + Flip flops
  - The combinational logic represents the transition relation



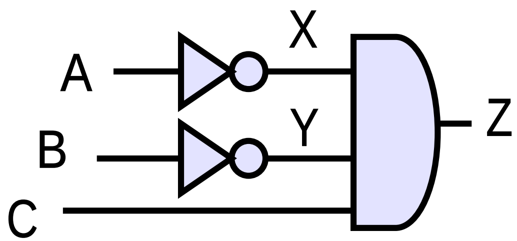


Transition Relation:

$g'_1 \Leftrightarrow r_1$

$g'_2 \Leftrightarrow \neg r_1 \wedge r_2 \wedge \neg g_1$

The notion of Characteristic Functions



x	y	c	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

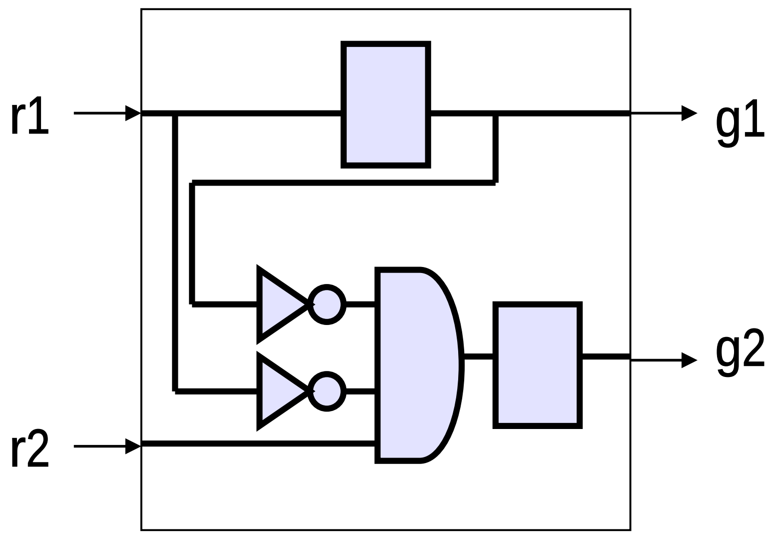
$f(z) = xyc$

The characteristic function $cf(z,x,y,c) \equiv (z = xyc)$
Therefore:

$$cf(z,x,y,c) = (z + \overline{x} + \overline{y} + \overline{c})(\overline{z} + x)(\overline{z} + y)(\overline{z} + c)$$

x	y	c	z	CF
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Characteristic functions for transition relations



Transition Relation:

$g'_1 \Leftrightarrow r_1$

$g'_2 \Leftrightarrow \neg r_1 \wedge r_2 \wedge \neg g_1$

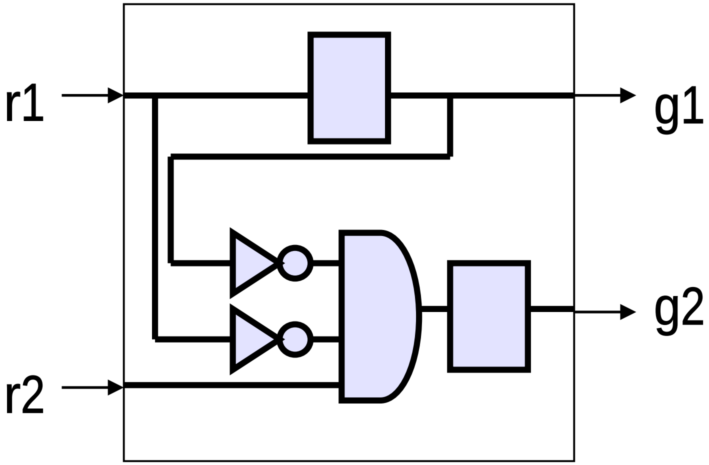
$$cf1(r_1, g'_1) = (\bar{r}_1 + g'_1)(r_1 + \bar{g}'_1)$$

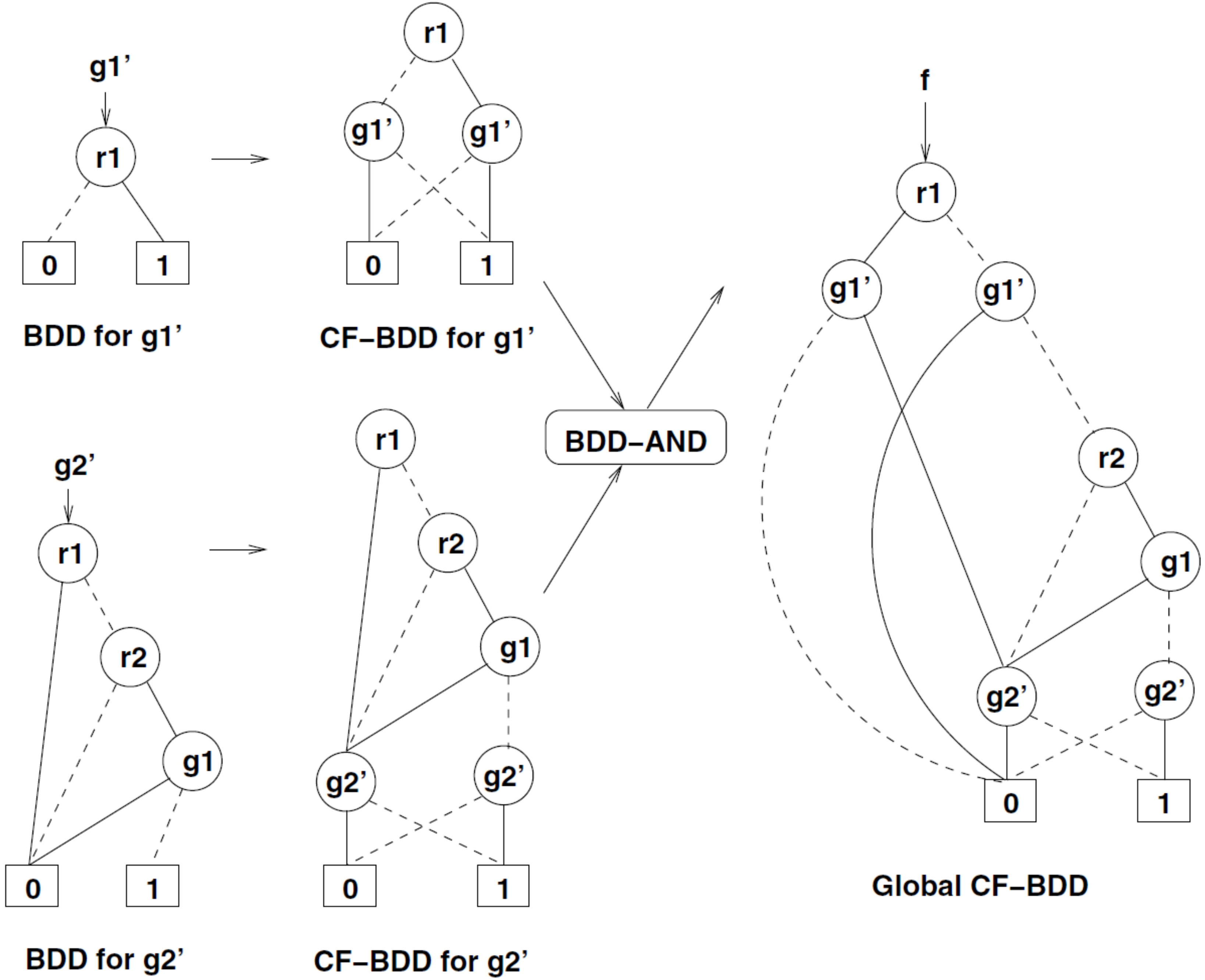
$$cf2(r_1, r_2, g_1, g'_2) = (g'_2 + r_1 + \bar{r}_2 + g_1)(\bar{g}'_2 + \bar{r}_1)\ (\bar{g}'_2 + r_2)(\bar{g}'_2 + \bar{g}_1)$$

$$cf(r_1, r_2, g_1, g_2, g'_1, g'_2) = cf1(r_1, g'_1) \wedge cf2(r_1, r_2, g_1)$$

$$= (\bar{r}_1 + g'_1)(r_1 + \bar{g}'_1)(g'_2 + r_1 + \bar{r}_2 + g_1)(\bar{g}'_2 + \bar{r}_1)\ (\bar{g}'_2 + r_2)(\bar{g}'_2 + \bar{g}_1)$$

Using BDDs





Transition Relation:

$g'_1 \Leftrightarrow r_1$

$g'_2 \Leftrightarrow \neg r_1 \wedge r_2 \wedge \neg g_1$

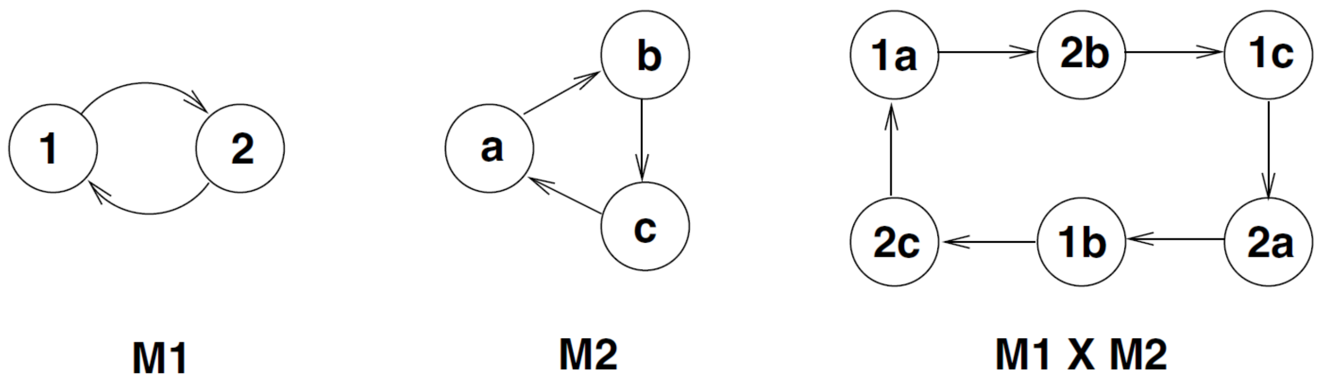
What can we do using CF of transition relation?

EXERCISE: Use the characteristic function for the transition relation to answer the following:

- Is there a transition from a state at which both requests, r1 and r2, are high to a state at which g2 is high?
- Can g1 ever be high for two consecutive cycles?
- Can g1 ever be high for three consecutive cycles?
- If g2 is high, does in mean r2 was high in one of the previous two cycles?

State Explosion and Succinct Representations

- The number of states in a circuit is a product of the number of states in its components (exponential growth)



- The size of BDDs grow exponentially with the number of variables.
  - There are model checking techniques which use *partitioned transition relations*
- The complexity of solving a SAT instance grows exponentially with the number of clauses.
  - But modern SAT solvers are good at solving millions of clauses in less than a second
- Techniques to overcome the state explosion problem
  - Abstractions, Assume-Guarantee, Induction