Binary Decision Diagrams (BDD)

Formal Systems (Spring 2014)

Dept. of Computer Science & Engg, IIT Kharagpur

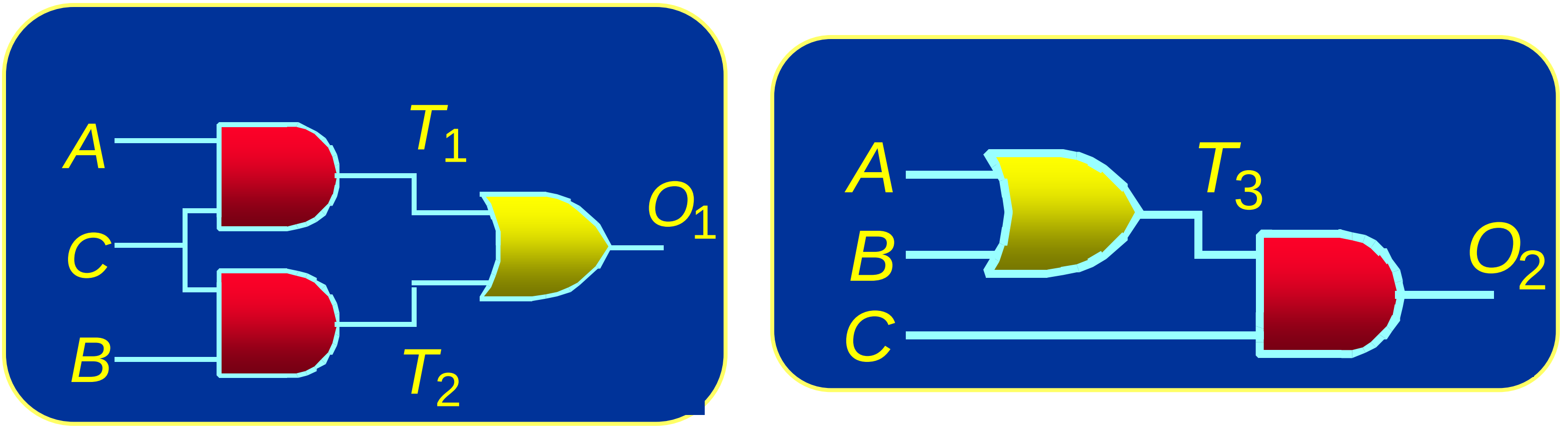
Aritra Hazra
Dept. of Computer Science & Engg.,
Indian Institute of Technology Kharagpur

Contents

- Motivation for Decision diagrams
- Binary Decision Diagrams
- ROBDD
- Effect of Variable Ordering on BDD size
- BDD operations
- Encoding state machines
- Reachability Analysis using OBDDs

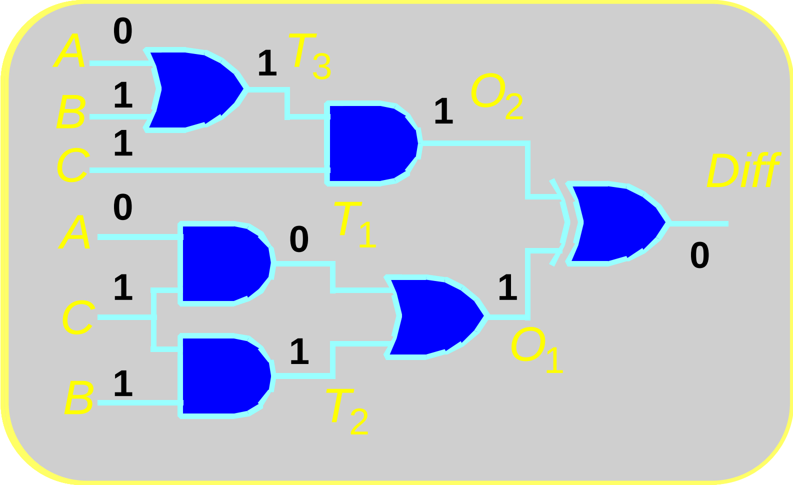
Sample Analysis Task

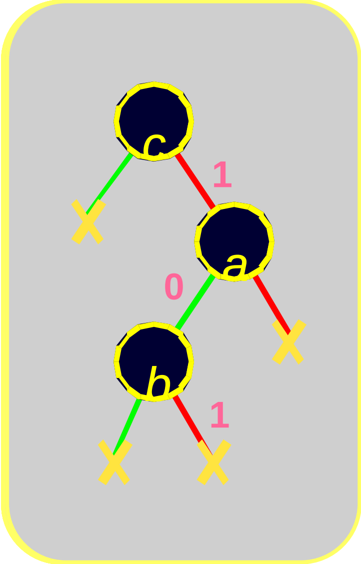
- **Logic Circuit Comparison**
  - **Do circuits compute identical function?**
    - **Basic task of formal hardware verification**
    - **Compare new design to "known good" design**



Solution by Combinatorial Search

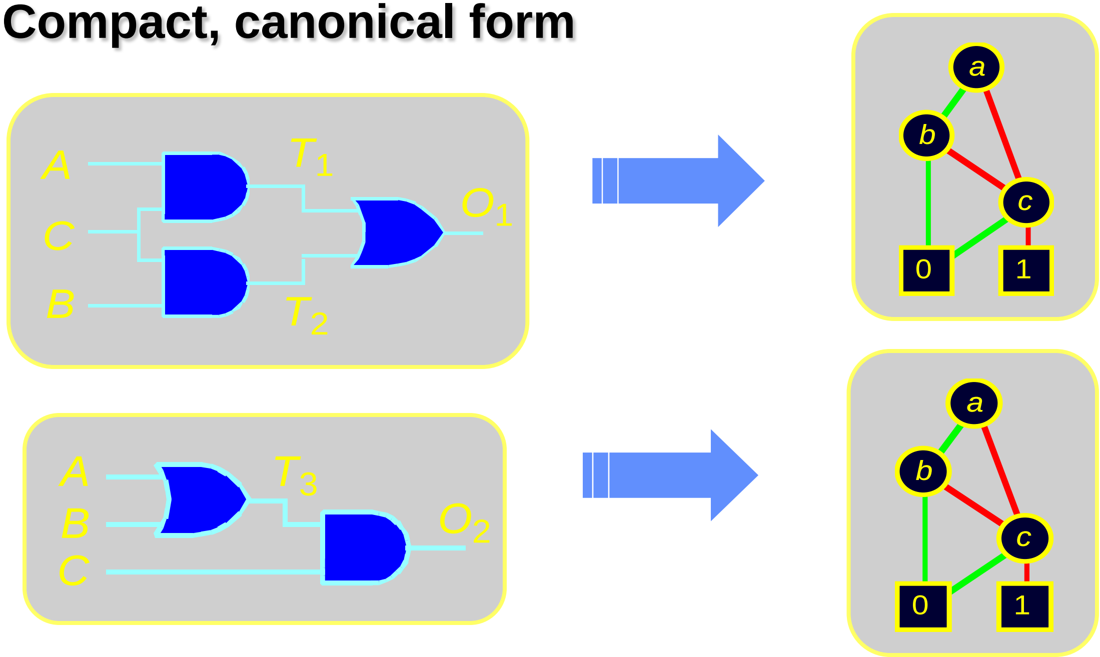
- **Satisfiability Formulation**
  - **Search for input assignment giving different outputs**
- **Branch & Bound**
  - **Assign input(s)**
  - **Propagate forced values**
  - **Backtrack when cannot succeed**
- **Challenge**
  - **Must prove all assignments fail**
  - **Typically explore significant fraction of inputs**
  - **Exponential time complexity**





Another Approach

- **Generate Complete Representation of Circuit Function**
  - Compact, canonical form



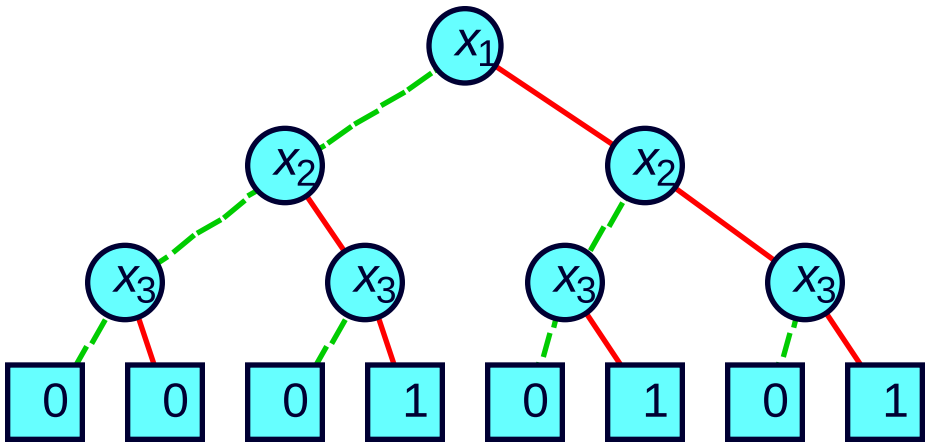
  - Functions equal if and only if representations identical
  - Never enumerate explicit function values
  - Exploit structure & regularity of circuit functions

Decision Structure

Truth Table

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Decision Tree



- Vertex represents decision
- Follow green (dashed) line for value 0
- Follow red (solid) line for value 1
- Function value determined by leaf value.

Binary Decision Diagram

- DAG representation of Boolean functions
- Operations on Boolean functions can be implemented as graph algorithms on BDDs
- Tasks in many problem domains can be expressed entirely in terms of BDDs
- BDDs have been useful in solving problems that would not be possible by more traditional techniques.

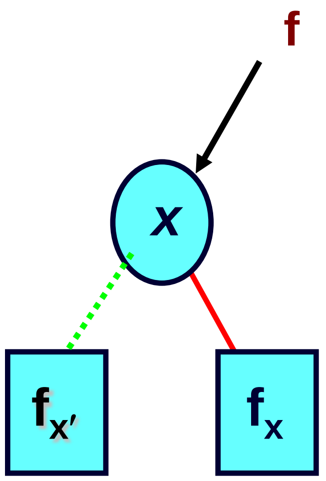
Binary Decision Diagram (BDD)

- **Each non-terminal vertex** v **is labeled by a variable** var(v) **and has arcs directed toward two children**
  - lo(v) **(dotted line) corresponding to the case where the variable is assigned 0**
  - hi(v) **(solid line) where the variable is assigned 1**
- **Each terminal vertex is labeled as 0 or 1**
- **For a given assignment to the variables, the value of the function is determined by tracing the path form root to a terminal vertex, following the branches appropriately**

BDDs and Shannon's Expansion

- Shannon's Expansion: $f = xf_x + x'f_{x'}$

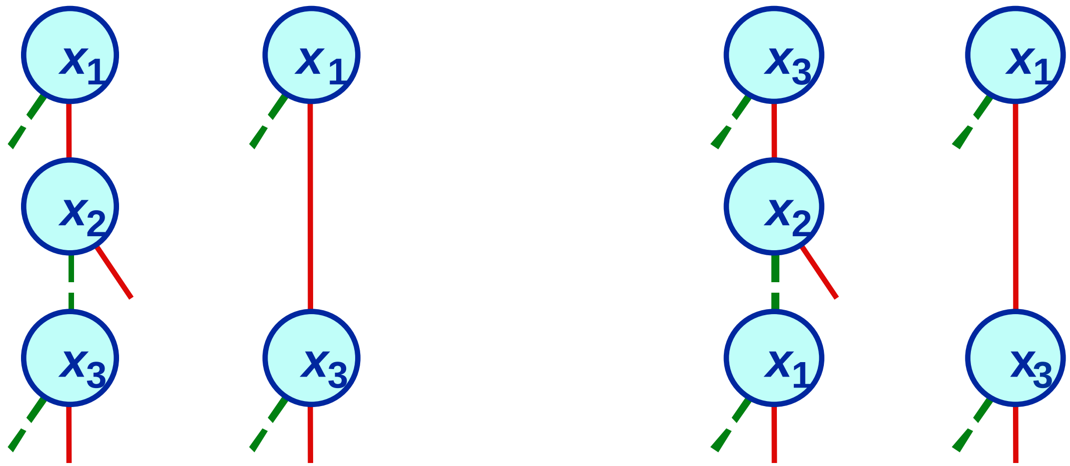
- BDD represents recursive application of Shannon's expansion



Ordered Binary Decision Diagram (OBDD)

- **Assign arbitrary total ordering to variables**
  - **e.g. $x_1 < x_2 < x_3$**
- **Variables must appear in ascending order along all paths**

OK



Not OK

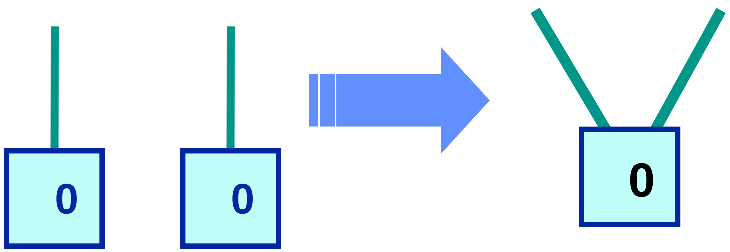
Properties

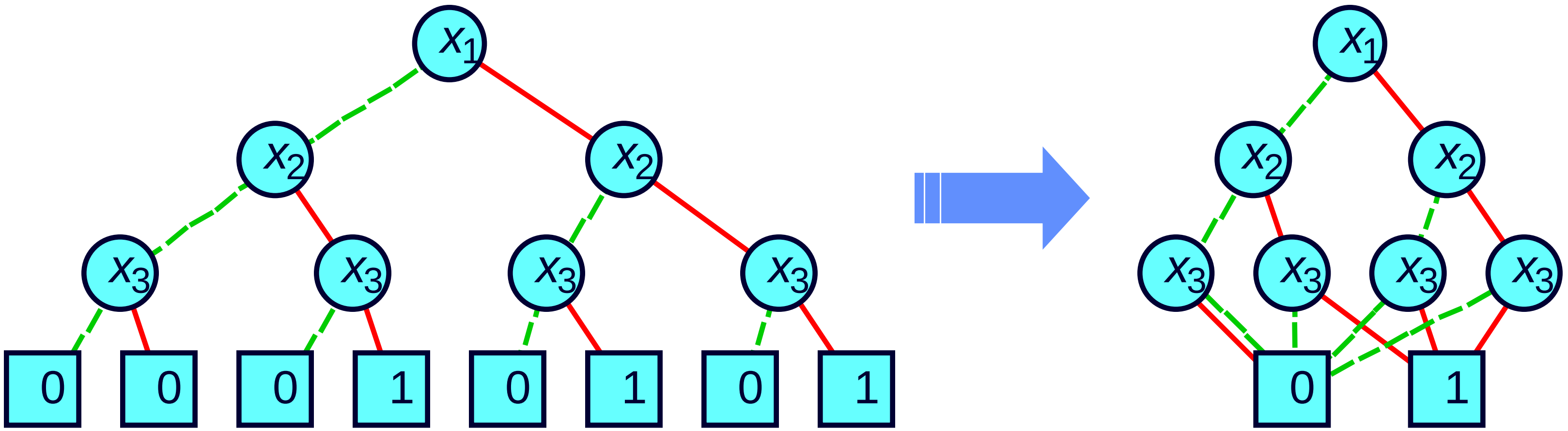
- No conflicting variable assignments along path
- Simplifies manipulation

Reduction Rule #1

Eliminate all but one terminal vertex with a given label and redirect all arcs into the eliminated vertices to the remaining

Merge equivalent leaves

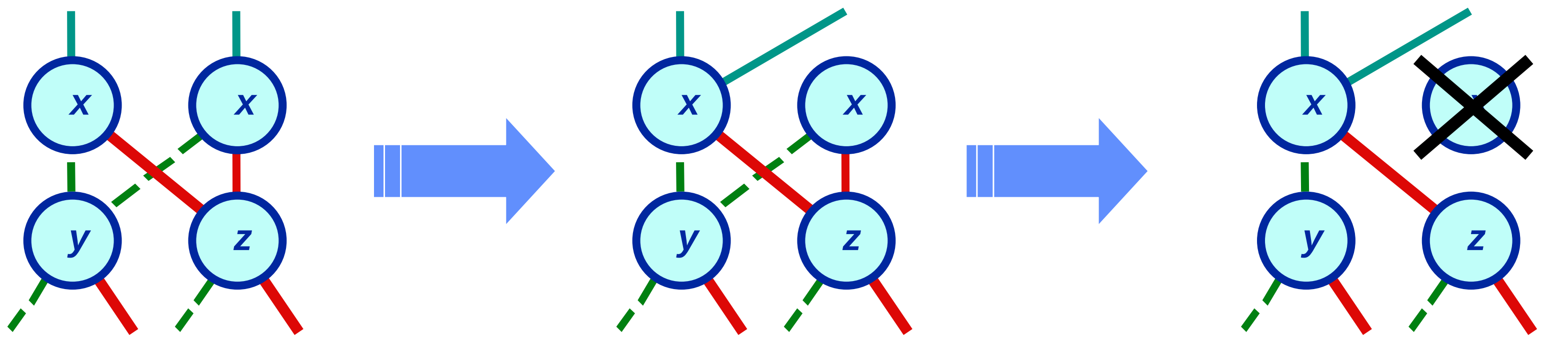


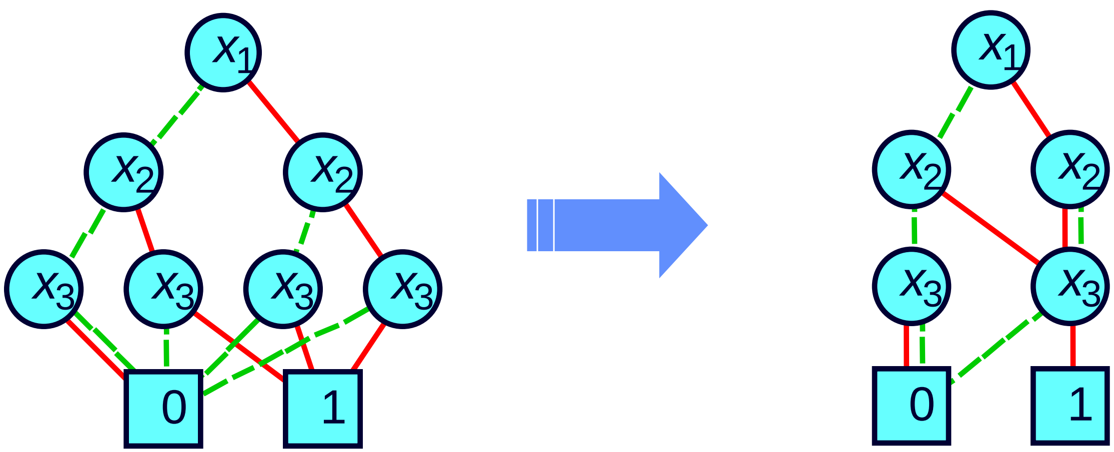


Reduction Rule #2

If non-terminal vertices u and v have var(u) = var(v), lo(u) = lo(v) and hi(u) = hi(v), eliminate one of them and redirect all incoming arcs to the other

Merge isomorphic nodes

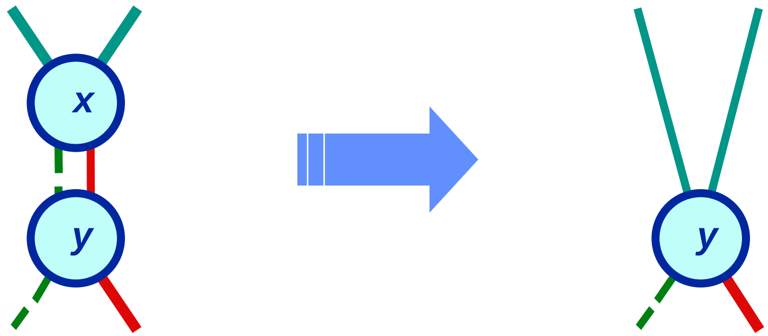


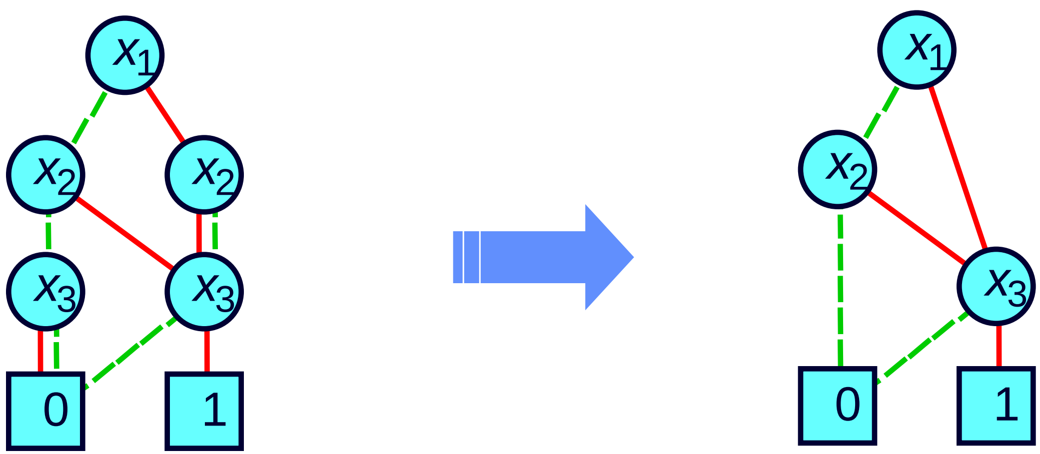


Reduction Rule #3

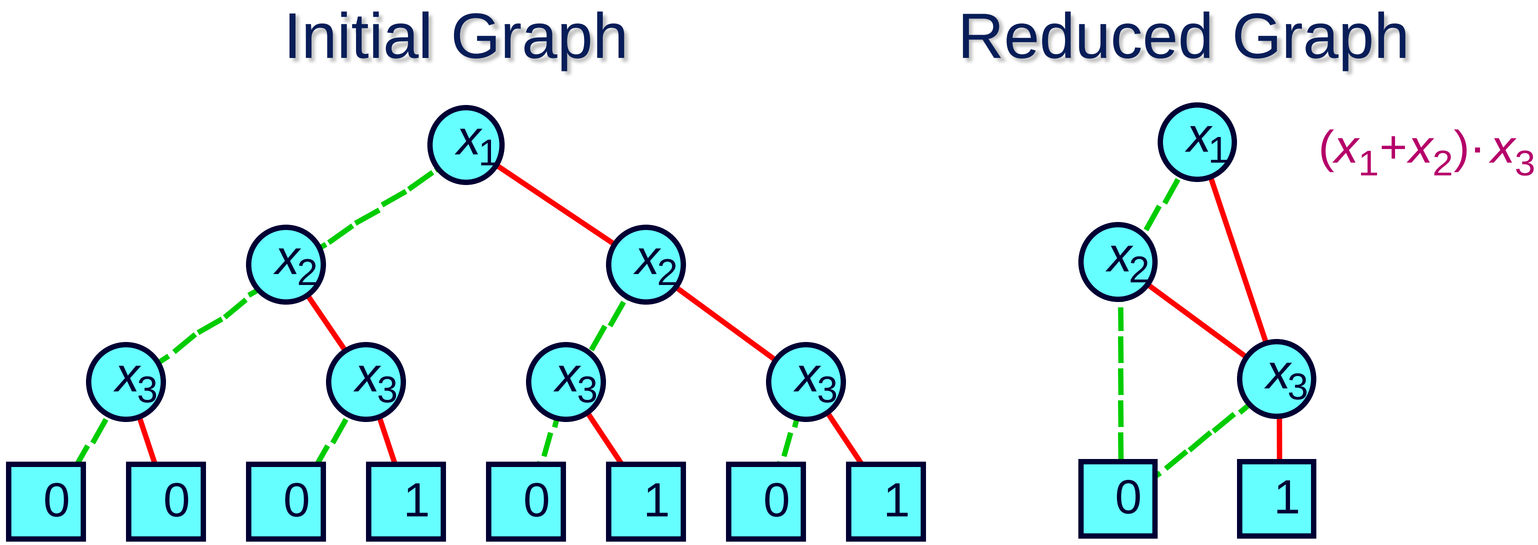
If non-terminal vertex v has lo(v) = hi(v), eliminate v and redirect all incoming arcs to lo(v)

Eliminate Redundant Tests





Reduced OBDD (ROBDD)



- Canonical representation of Boolean function
- For the same variable ordering, two functions equivalent if and only if graphs isomorphic
  - Can be tested in linear time

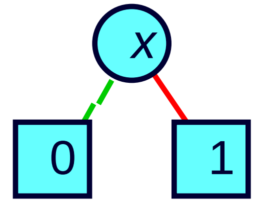
Some Example Functions

Constants

0 **Unique unsatisfiable function**

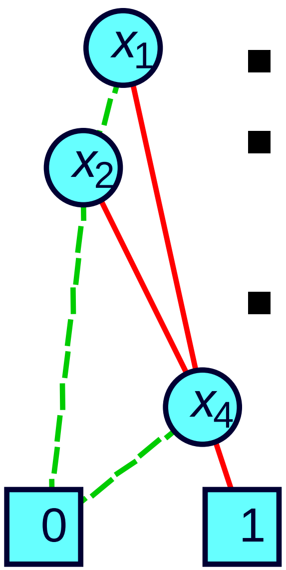
1 **Unique tautology**

Variable



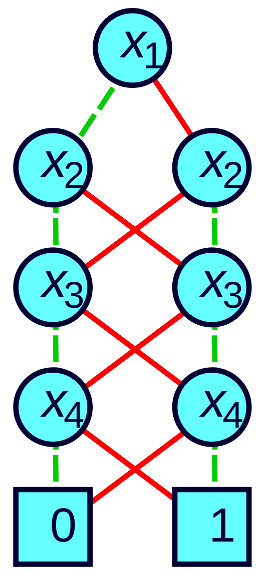
Treat variable as function

Typical Function



- $(x_1 \vee x_2) \wedge x_4$
- **No vertex labeled x_3**
  - independent of x_3
- **Many subgraphs shared**

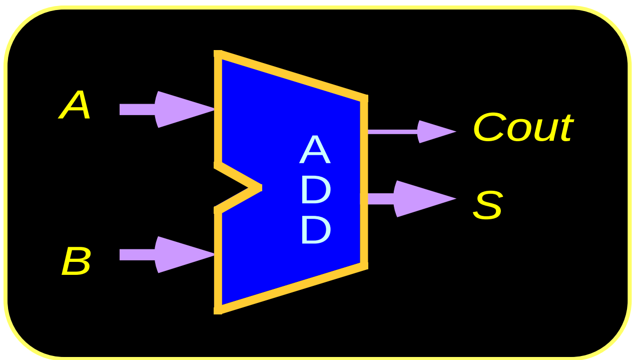
Odd Parity



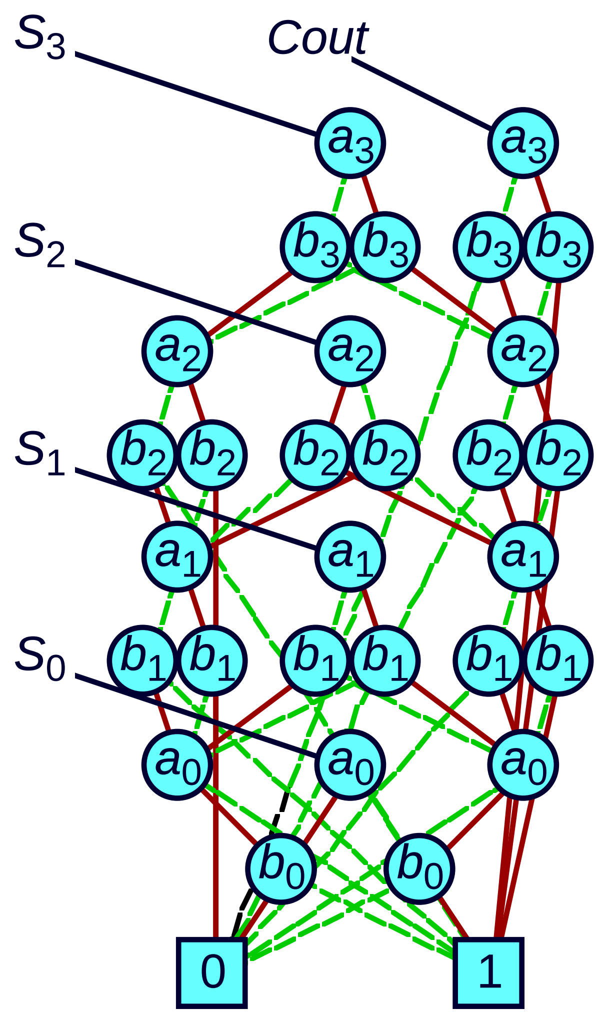
Linear representation

Circuit Functions

- **Functions**
  - All outputs of 4-bit adder
  - Functions of data inputs



- **Shared Representation**
  - Graph with multiple roots
  - 31 nodes for 4-bit adder
  - 571 nodes for 64-bit adder
  - Linear Growth

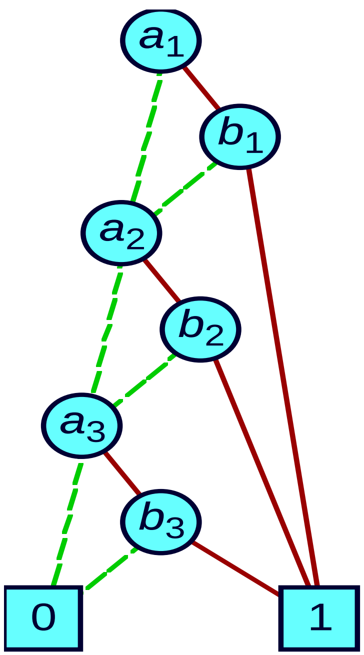


Effect of Variable Ordering on ROBDD Size

$$(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$$

Good Ordering

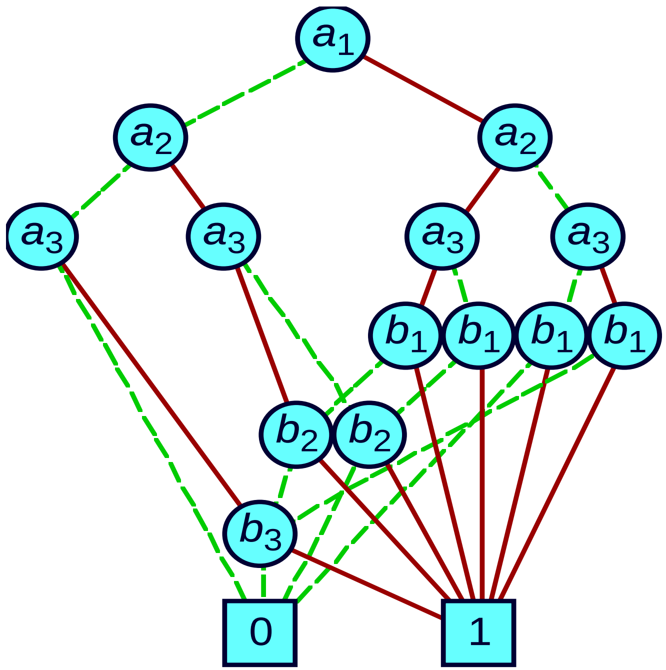
($a_1 < b_1 < a_2 < b_2 < a_3 < b_3$)



Linear Growth

Bad Ordering

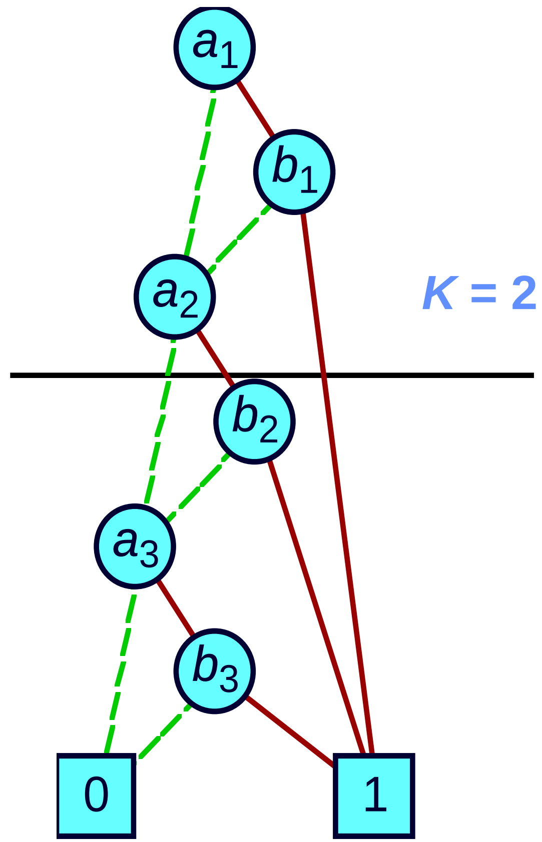
($a_1 < a_2 < a_3 < b_1 < b_2 < b_3$)

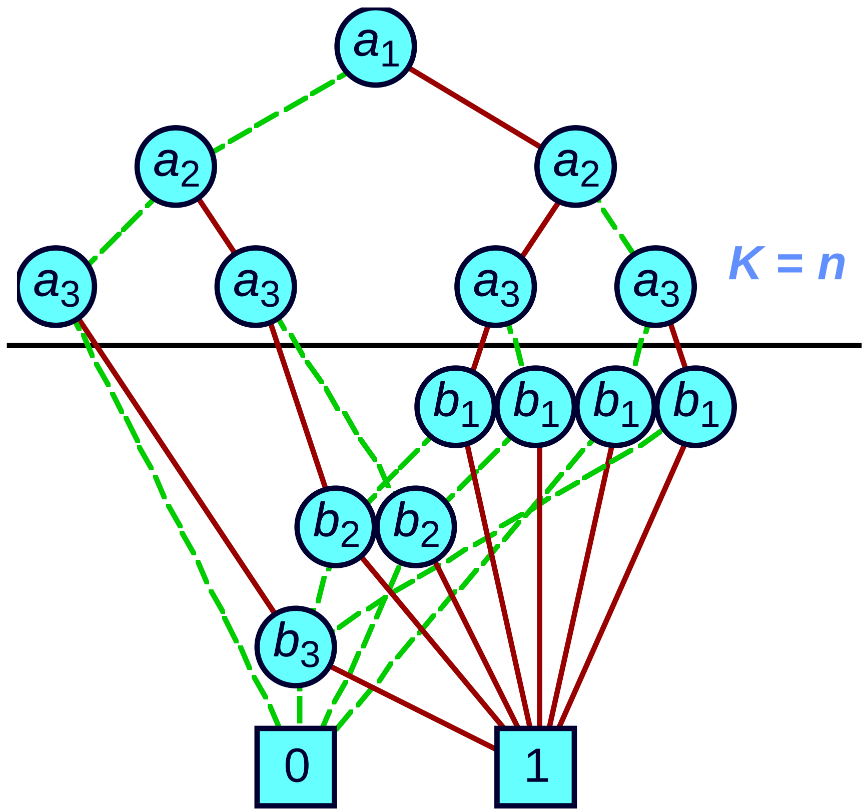


Exponential Growth

Analysis of Ordering Example

$$(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$$





Selecting a good Variable Ordering

- **Intractable Problem**
  - Even when problem represented as OBDD
- **A good variable ordering should use**
  - Local computability
  - Ordering based on power to control output
- **Application-Based Heuristics**
  - Exploit characteristics of application
    - Ordering for functions of combinational circuit
    - Traverse circuit graph depth-first from outputs to inputs
    - Assign variables to primary inputs in order encountered

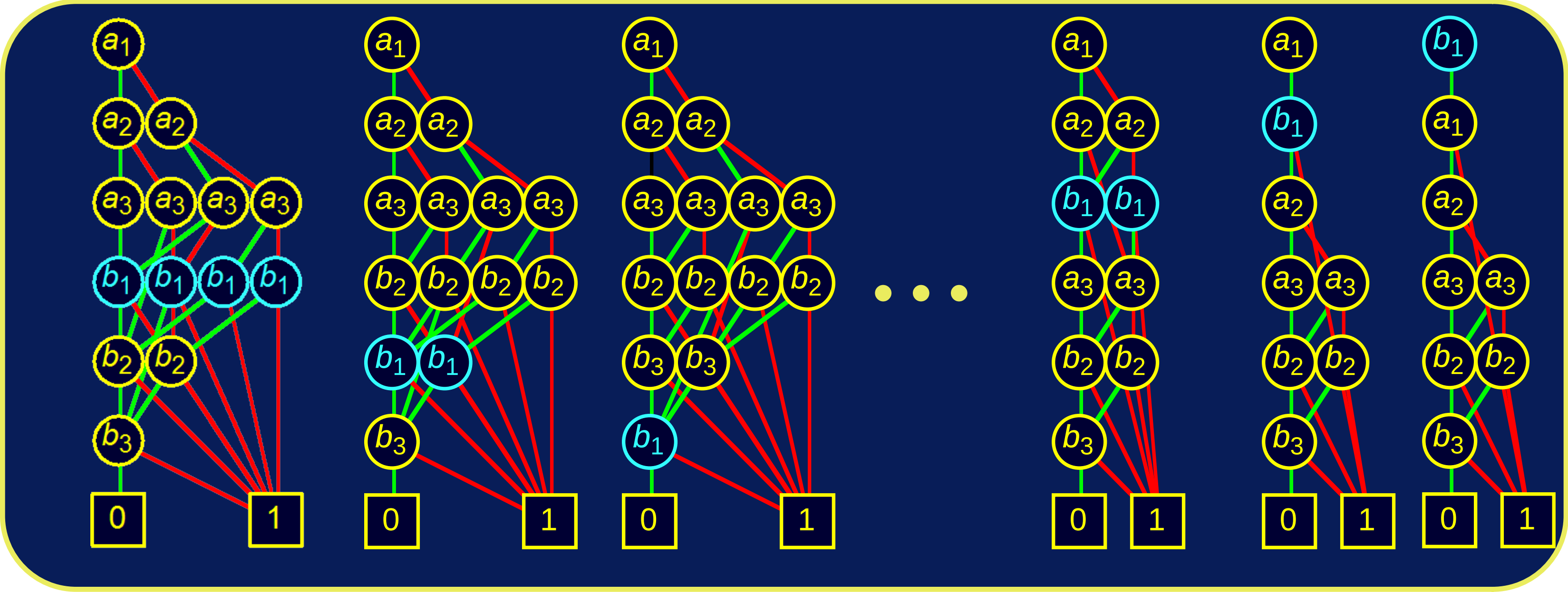
Dynamic Variable Ordering

- Rudell, ICCAD ‘93
- Concept
  - Variable ordering changes as computation progresses
    - Typical application involves long series of BDD operations
  - Proceeds in background, invisible to user
- Implementation
  - When approach memory limit, attempt to reduce
    - Garbage collect unneeded nodes
    - Attempt to find better order for variables
  - Simple, greedy reordering heuristics

Dynamic Reordering By Sifting

- Choose candidate variable
- Try all positions in ordering
  - Repeatedly swap with adjacent variable
- Move to best position found

Best Choices



Sample Function Classes

Function Class	Best	Worst	Ordering Sensitivity
ALU (Add/Sub)	linear	exponential	High
Symmetric	linear	quadratic	None
Multiplication	exponential	exponential	Low

- **General Experience**
  - **Many tasks have reasonable OBDD representations**
  - **Algorithms remain practical for up to 100,000 node OBDDs**
  - **Heuristic ordering methods generally satisfactory**

BDD Operations

- **Strategy**
  - Represent data as set of OBDDs
    - Identical variable orderings
  - Express solution method as sequence of symbolic operations
  - Implement each operation by OBDD manipulation
- **Algorithmic Properties**
  - Arguments are OBDDs with identical variable orderings.
  - Result is OBDD with same ordering.
  - “Closure Property”

The APPLY Operation

- Given argument functions f and g, and a binary operator <op>, APPLY returns the function f <op> g

- Works by traversing the argument graphs depth first

- Algebraic operations "commute" with the Shannon expansion for any variable x
  - $f <op> g = x' (f|_{x=0} <op> g|_{x=0}) + x ((f|_{x=1} <op> g|_{x=1})$

The Apply Algorithm

- Consider a function f represented by a BDD with root vertex r_f
- The restriction of f with respect to a variable x such that

  $x \leq var(r_f)$ can be computed as :

$$
\begin{aligned}
f\,|_{x=b} &= r_f\,, && x < var(r_f) \\
&= lo(r_f), && x = var\,(r_f) \text{ and } b = 0 \\
&= hi(r_f), && x = var\,(r_f) \text{ and } b = 1
\end{aligned}
$$

- The algorithm for APPLY utilizes the above restriction definition.

The Apply Algorithm

- Each evaluation step is identified by a vertex from each of the argument graphs
- Suppose functions f and g are represented by root vertices r_f and r_g
- If r_f and r_g are both terminal vertices, terminate and return an appropriately labeled terminal vertex e.g. (A_4, B_3) and (A_5, B_4)

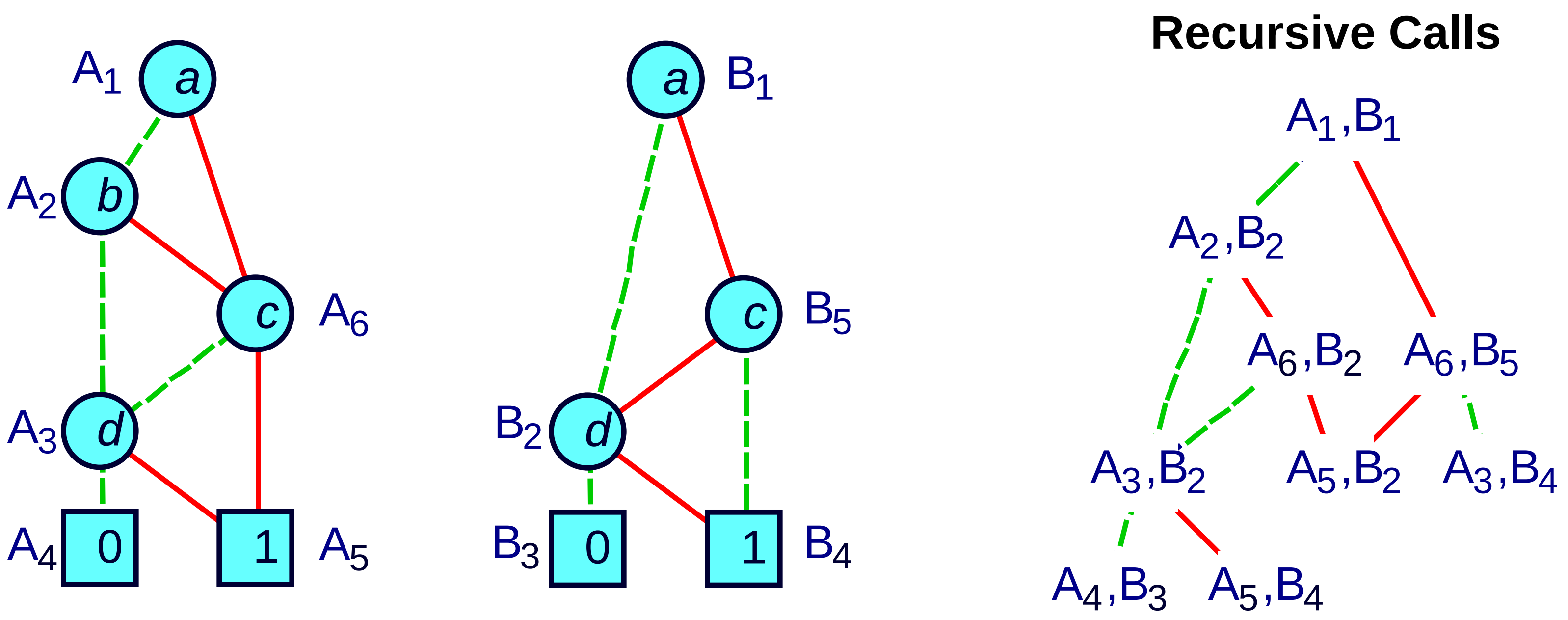
The Apply algorithm

- Let x be the splitting variable

  ($x = \min(var(r_f), var(r_g))$)

- BDDs for ($f|_{x=0}$ <op> $g|_{x=0}$) and ($f|_{x=1}$ <op> $g|_{x=1}$) are computed by recursively evaluating the restrictions of f and g for value 0 and for value 1

Example



- Initial evaluation with vertices A_1, B_1 causes recursive evaluations with vertices A_2, B_2 and A_6, B_5

Apply operation

- Reaching a terminal with a dominant value (e.g 1 for OR, 0 for AND) terminates recursion and returns an appropriately labeled terminal (A_5, B_2 and A_3, B_4)

- Avoid multiple recursive calls on the same pair of arguments by a hash table (A_3, B_2 and A_5, B_2)

Apply operation

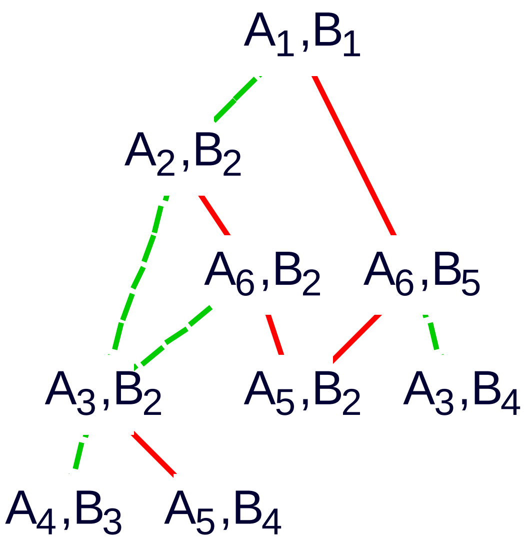
- Each evaluation step returns a vertex in the generated graph

- Apply reduction before merging the result

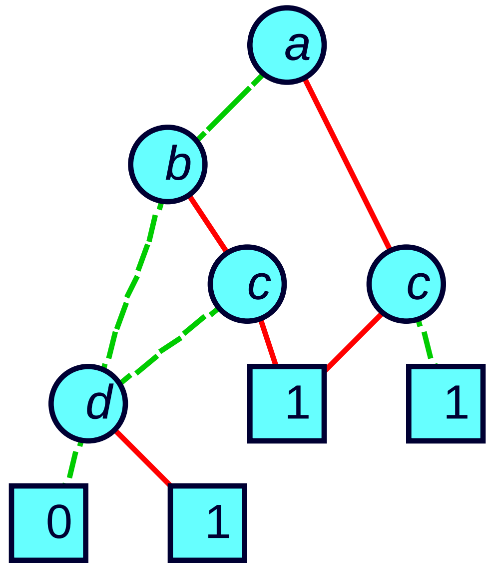
- Complexity of operation : $O(m_f * m_g)$ where m_f and m_g represent the number of vertices in the BDDs for f and g respectively

Example

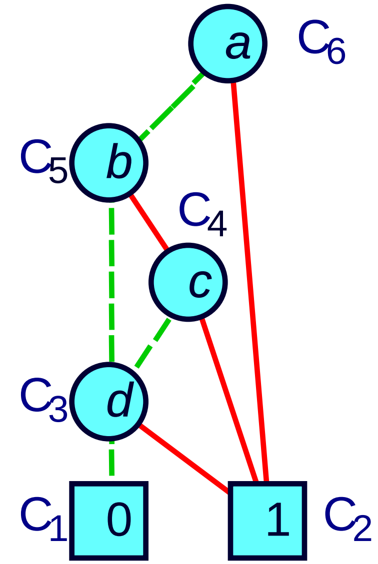
Recursive Calls



Without Reduction



With Reduction

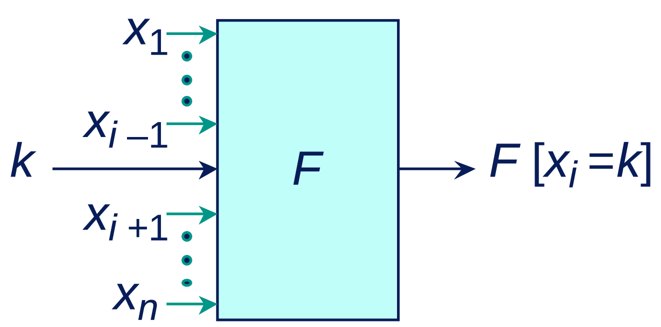


Restrict Operation

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

F_x equivalent to $F\,[x = 1]$

$F_{\overline{x}}$ equivalent to $F\,[x = 0]$



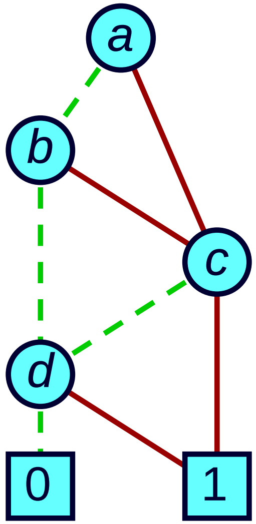
Restriction Algorithm

Implementation

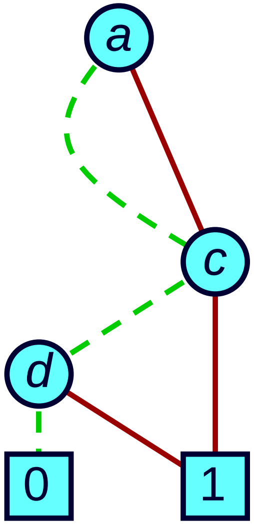
- Depth-first traversal
- Redirect any arc into vertex v having var(v) = x to point to hi(v) for x =1 and lo(v) for x = 0
- Complexity linear in argument graph size

Restriction Execution Example

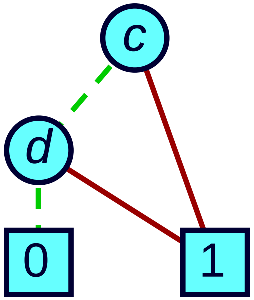
Argument *F*



Restriction *F*[*b*=1]



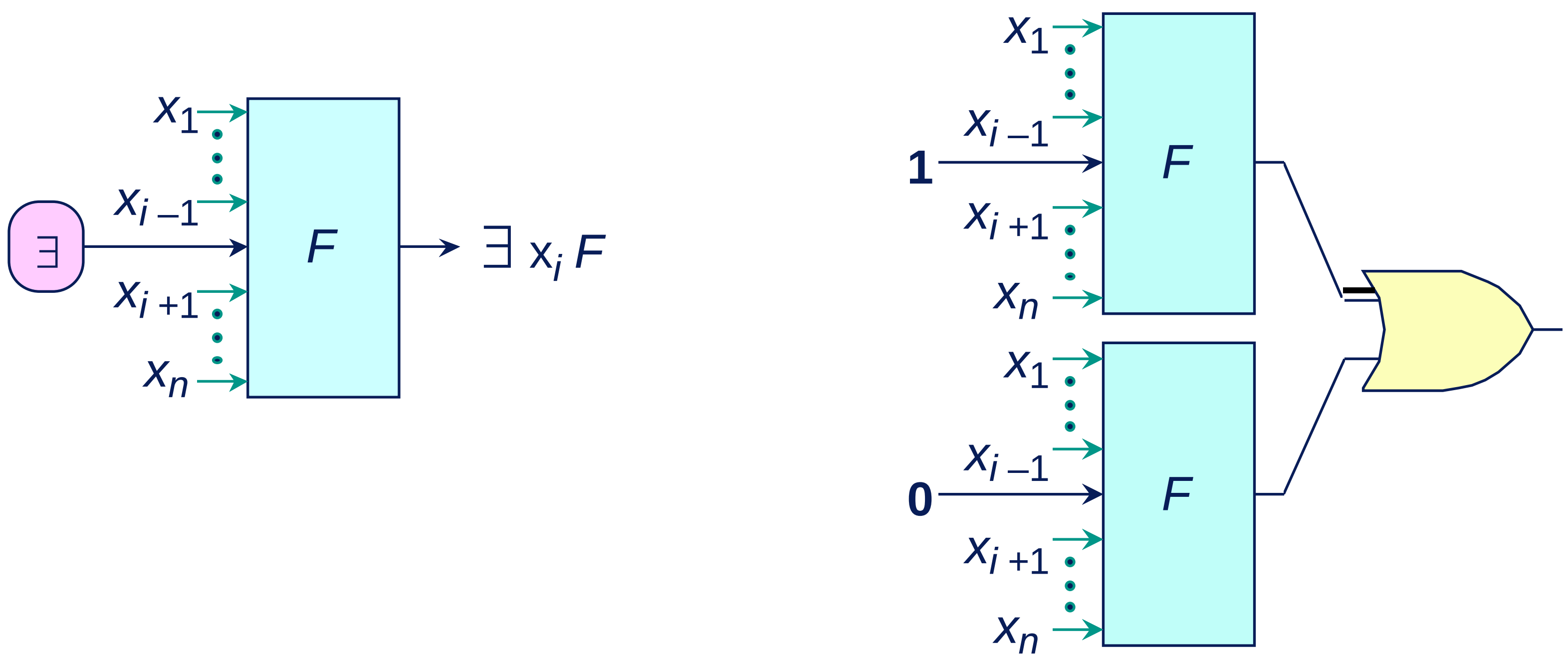
Reduced Result



Derived Operations

- Express as combination of Apply and Restrict
- Preserve closure property
  - Result is an OBDD with the right variable ordering
- Polynomial complexity
  - Although can sometimes improve with special implementations

Variable Quantification



- **Eliminate dependency on some argument through quantification**
- **Combine with AND for universal quantification.**

Digital Applications of BDDs

- **Verification**
  - Combinational equivalence (UCB, Fujitsu, Synopsys, …)
  - FSM equivalence (Bull, UCB, MCC,Colorado, Torino, …)
  - Symbolic Simulation (CMU, Utah)
  - Symbolic Model Checking (CMU, Bull, Motorola, …)
- **Synthesis**
  - Don’t care set representation (UCB, Fujitsu, …)
  - State minimization (UCB)
  - Sum-of-Products minimization (UCB, Synopsys, NTT)
- **Test**
  - False path identification (TI)

Some Popular BDD packages

- CUDD (Colorado University Decision Diagram)
- TUD BDD package (TUDD)
- BUDDY
- CMU BDD

Informations about the above BDD packages and some more details can be found at http://www.bdd-portal.org/

Finite State System Analysis

- Systems Represented as Finite State Machines
  - Analysis Tasks
  - State reachability
  - State machine comparison
  - Temporal logic model checking
- Traditional Methods Impractical for Large Machines
  - Polynomial in number of states
  - Number of states exponential in number of state variables.
  - Example: single 32-bit register has 4,294,967,296 states!

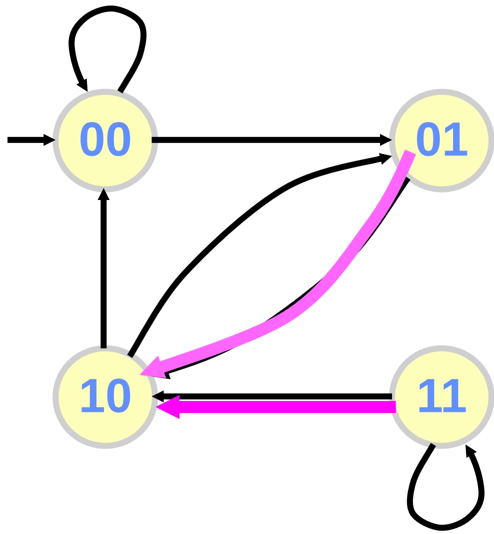
Symbolic FSM Representation

- **Represent set of transitions as function δ(*Old*, *New*)**
  - **Yields 1 if can have transition from state *Old* to state *New***

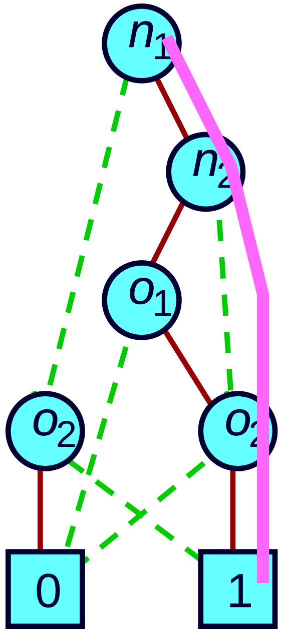
- **Represent as Boolean function**
  - **Use variables for encoding states**

Symbolic FSM Representation

Nondeterministic FSM



Symbolic Representation

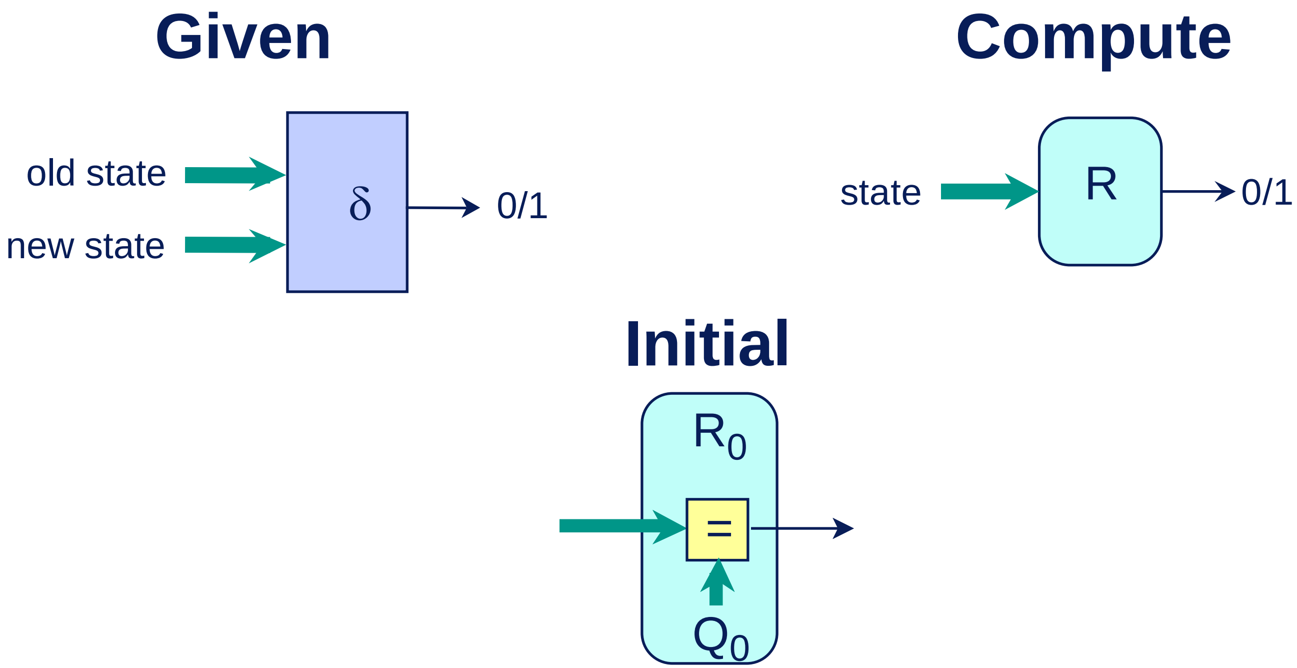


o_1, o_2 encoded old state

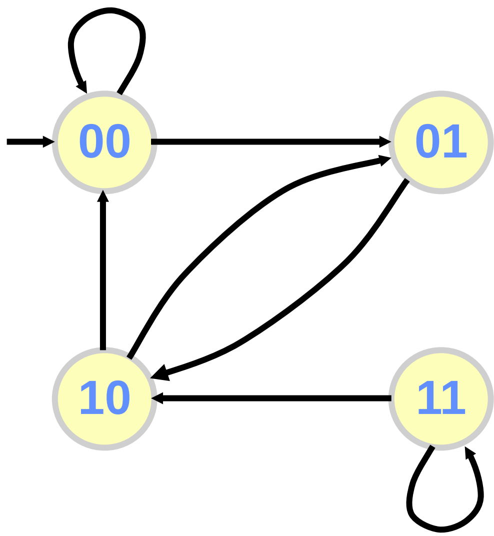
n_1, n_2 encoded new state

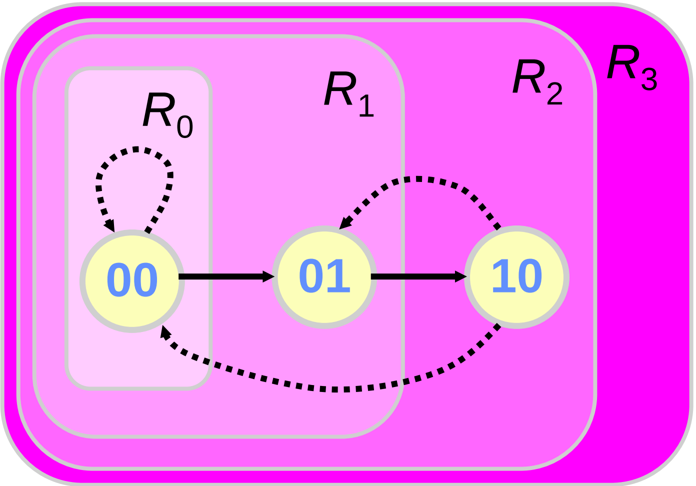
Reachability Analysis

- **Compute set of states reachable from initial state (Q_0 = 00)**
- **Represent as Boolean function R(*S*)**



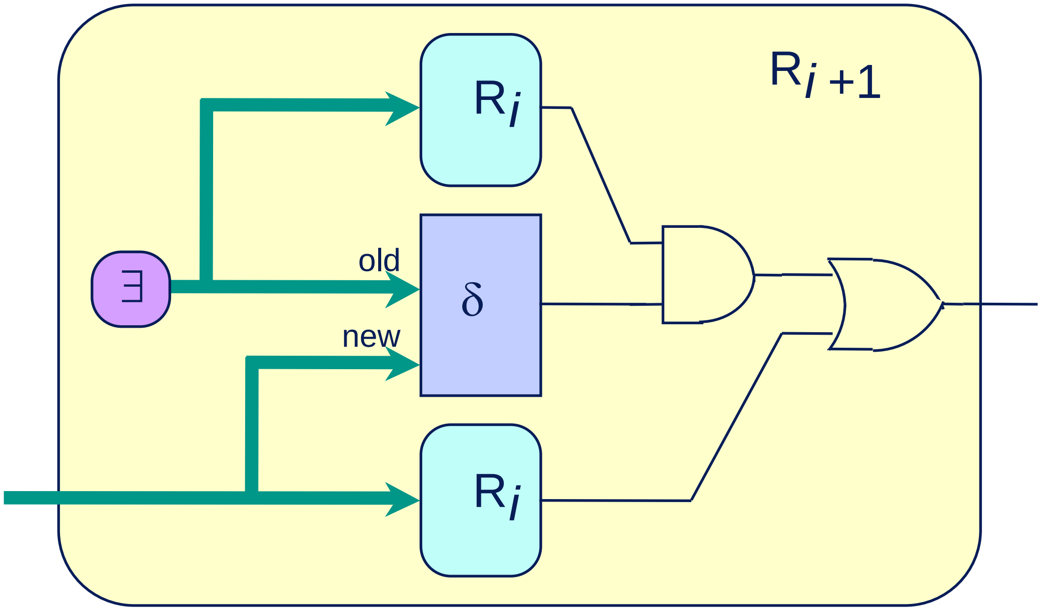
Breadth-First Reachability Analysis





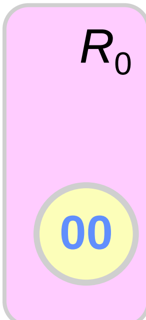
- R_i – set of states that can be reached in i transitions
- Reach fixed point when $R_n = R_{n+1}$
  - Guaranteed since finite state

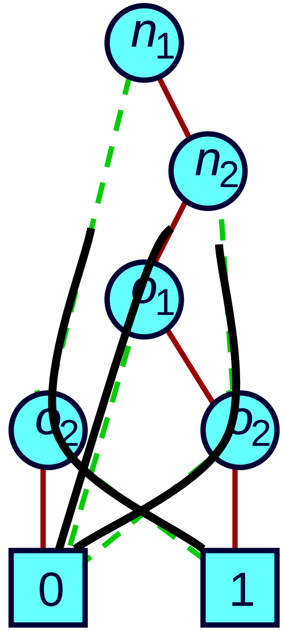
Iterative Computation

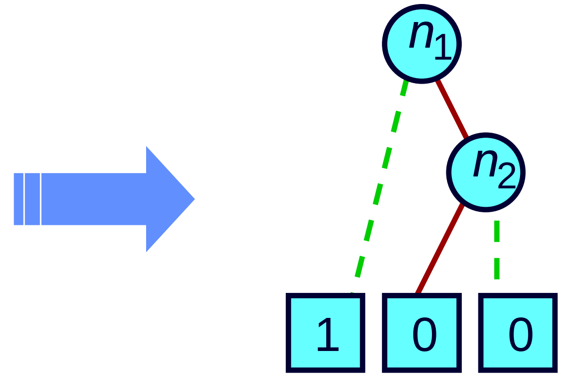


- R_{i+1} – set of states that can be reached within $i+1$ transitions
  - Either in R_i
  - or single transition away from some element of R_i

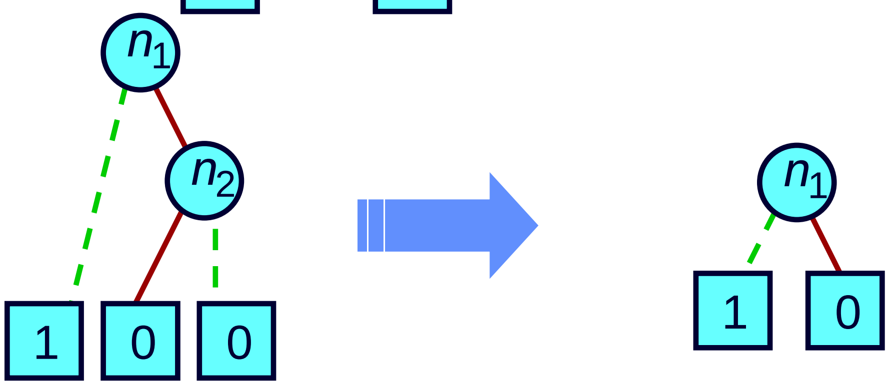
Example: Computing R_1 from R_0

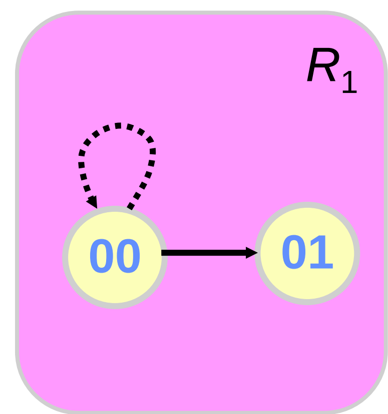






$$\exists\, Old\ [R_0(Old) \wedge \delta(Old, New)]$$





What's good about OBDDs ?

- **Powerful Operations**
  - Creating, manipulating, testing
  - Each step polynomial complexity
    - Graceful degradation
  - Maintain "closure" property
    - Each operation produces form suitable for further operations
- **Generally Stay Small Enough**
  - Especially for digital circuit applications
  - Given good choice of variable ordering
- **Weak Competition**

What's not good about OBDDs?

- **Doesn't Solve All Problems**
  - Can't do much with multipliers
  - Some problems just too big
  - Weak for search problems

- **Must be Careful**
  - Choose good variable ordering
  - Some operations too hard

Zero Suppressed BDD's - ZBDD's

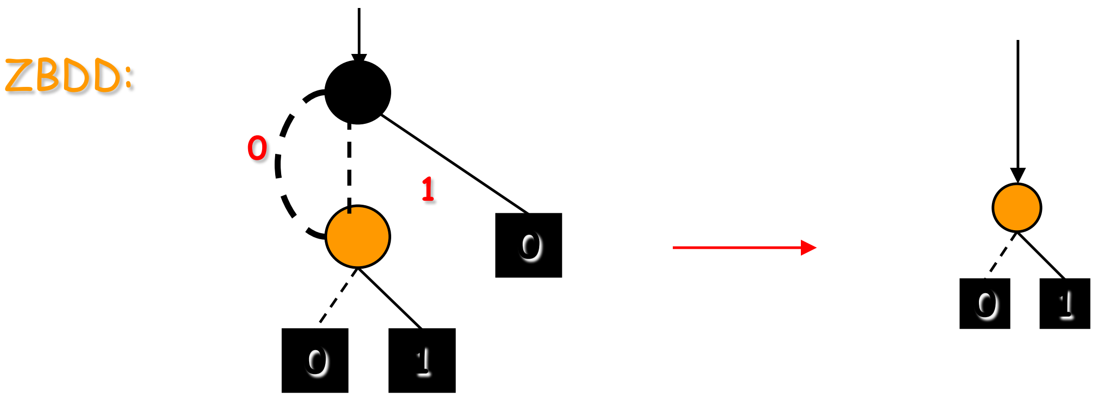
- ZBDD's were invented by Minato to efficiently represent sparse sets. They have turned out to be extremely useful in implicit methods for representing primes (which usually are a sparse subset of all cubes).

- Different reduction rules.

Zero Suppressed BDD's - ZBDD's

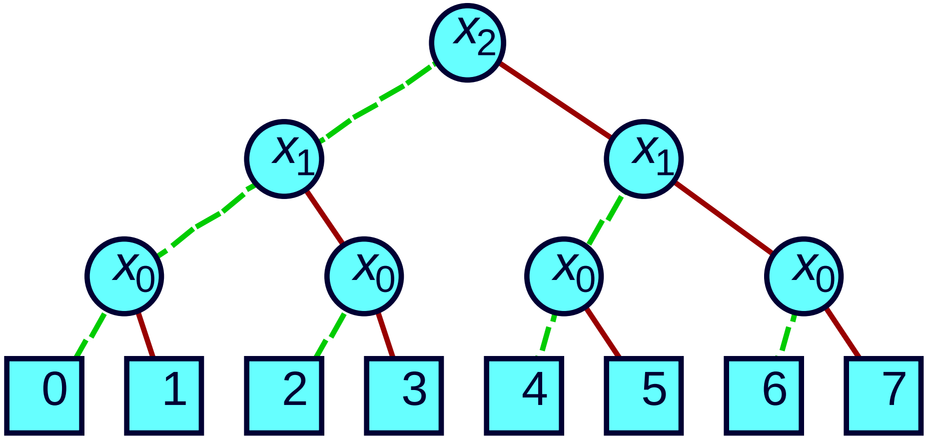
- **ZBDD Reduction Rule:: eliminate all nodes where the then node points to 0. Connect incoming edges to else node**

- **For ZBDD, equivalent nodes can be shared as in case of BDDs.**



MTBDD- Multiterminal BDD

$$x_0 + 2x_1 + 4x_2$$

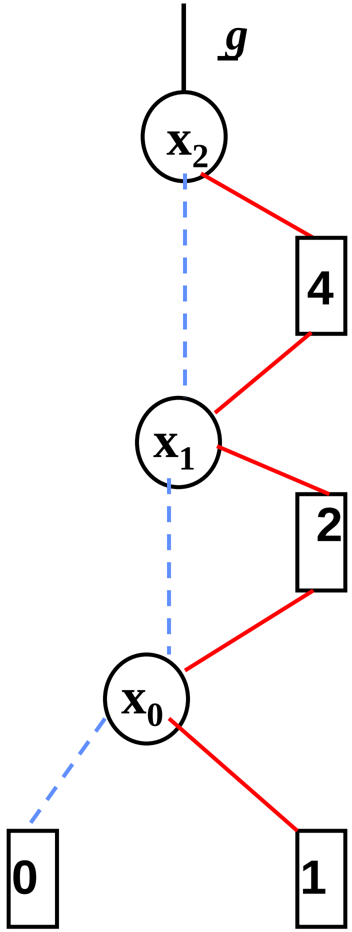


- Evaluating a MTBDD for a given variable assignment is similar to that in case of BDD
- Very inefficient for representing functions yielding values over a large range

EVBDD – Edge value BDD

- EVBDDs can be used when the number of possible function values are too high for MTBDDs.

- Evaluating a EVBDD involves tracing a path determined by the variable assignment, summing the weights and the terminal node value



*BMD(Binary Moment Diagrams)

- **Features**
  - Used for Word level simulation/verification
  - Canonical
  - Based on linear decomposition of a function

- **Functional Decomposition :**

$$f = (1-x)\, f_{\sim x} + (x)\, f_x$$

$$= f_{\sim x} + x\, (\, f_x - f_{\sim x})$$

$$= f_{\sim x} + x\, (\, f_{\cdot x}\,)$$

where $f_{\cdot x}$ is the linear moment w.r.t. x

Representing *BMDs

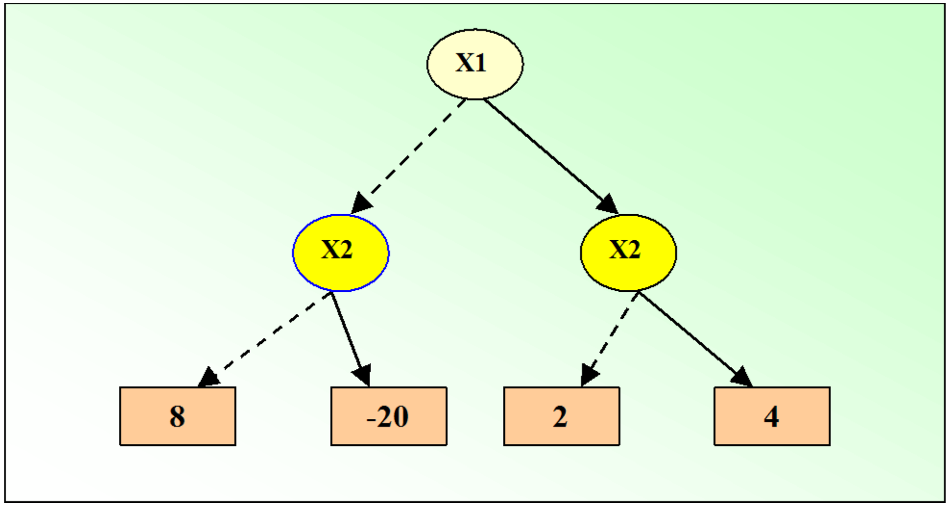
- **Graph :**
  - Example

f = (1-x1)(1-x2)(8)+(1-x1)(x2)(-12)

+(x1)(1-x2)(10) + (x1)(x2)(-6)

= 8 - 20(x2) + 2(x1) + 4(x1*x2)

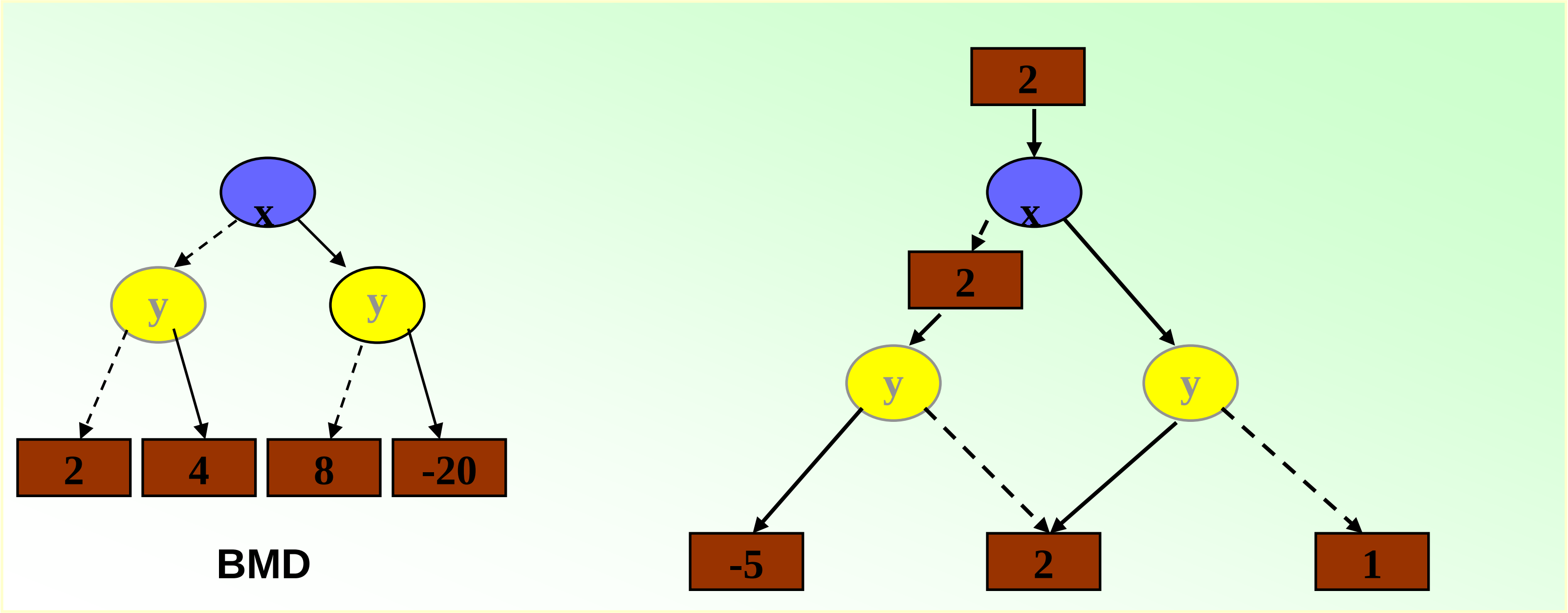
x1	x2	f
0	0	8
0	1	-12
1	0	10
1	1	-6



Edge Weights (*BMDs)

Weights combine multiplicatively along path from root to leaf Rules :

- weights of 2 branches relatively prime
- weight 0 allowed only for terminal vertices
- if one edge has weight 0, the other has weight 1



* BMD

References

- **Graph Based Algorithms for Boolean Function Manipulation,**

  Randal E. Bryant, IEEE Transactions on Computers, Volume: C-35, Issue: 8, pp. 677-691, August 1986.

- **Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams,**

  Randal E. Bryant, ACM Computing Surveys, Volume: 24 Issue: 3, pp. 293-318, September 1992.

- **An Introduction to Binary Decision Diagrams,**

  Henrik Reif Andersen, Technical Report, Course Notes on the WWW, October 1997.