

# Boolean Functions And Their Representations by Binary Decision Diagrams

## Boolean Functions

- Definition
  - A map that associates a Boolean value to each possible assignment of input variables
- Widely Used in CAD Applications
  - Hardware synthesis
  - Test generation
  - Validation
- Uniform Description in Model Checking
  - Describing the model (Transition Relation)
  - Representing the Set of States

## Representing Boolean Functions

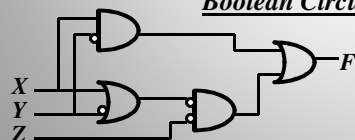
- Truth Table
  - Function's value for each assignment of inputs
- Boolean Expressions
  - Formulas using basic Boolean operations
- Boolean Circuits
  - Network of logical gates
- Binary Decision Trees
  - Each path in a graph represents one assignment of input variables

## Representing Boolean Functions: Examples

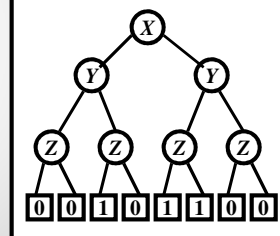
***Truth Table***

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

***Boolean Circuit***



***Binary Decision Tree***



***Boolean Expression***

$$F = \neg X \wedge Y \wedge \neg Z \vee X \wedge \neg Y$$

## Good Representation: Requirements

- Compactness
  - Grow not too fast with number of inputs
- Efficient Logic Manipulation
  - Computing Boolean functions
  - Performing equivalence check
  - Finding Out the satisfying assignment
    - What are the values of input variables that evaluate the Boolean function to "TRUE"

## Good Representation: Is It Possible?

- Theoretical Fact
  - No good representation exists for **all** Boolean functions
- The concern is with practically interesting cases
  - Human designed models are "regular"

## Binary Decision Trees (BDT) and Binary Decision Diagrams (BDD)

### Toward BDD: Key Ideas

- Use Binary Decision Diagrams
  - Two edges of a node of function  $F$  point to functions  $F(\mathbf{x}=\mathbf{0})$  and  $F(\mathbf{x}=\mathbf{1})$
  - Basis for recursive manipulation
- Fix the Order of Input Variables
  - On each path in Decision Diagram the order of variables is the same
  - A step to canonical form
- Share All Equal Functions
  - Efficient comparison

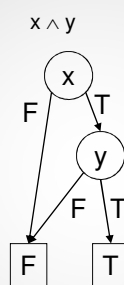
## Towards BDD: Key Ideas

- Represent Only Essential Variables
  - Don't put variables on paths where they cannot change the value of a function
- Reduction Rule
  - Delete a node whose both edges point to the same function
- Any Boolean formula  $f$  on variables  $x_1, x_2, \dots, x_n$  can be written as (called Shannon expansion):  
$$f = x_i \wedge f [\text{True}/x_i] \vee \neg x_i \wedge f [\text{False}/x_i] \quad (\text{this is an if-then-else})$$
- BDDs use this idea

## BDDs from Another Perspective

This node corresponds to the formula False, which comes from the Shannon expansion:

$$\text{False} = x \wedge y [\text{False}/x]$$



This node corresponds to the formula  $y$ , which comes from the Shannon expansion:  
$$y = x \wedge y [\text{True}/x]$$

## Reduced and Ordered Binary Decision Diagrams

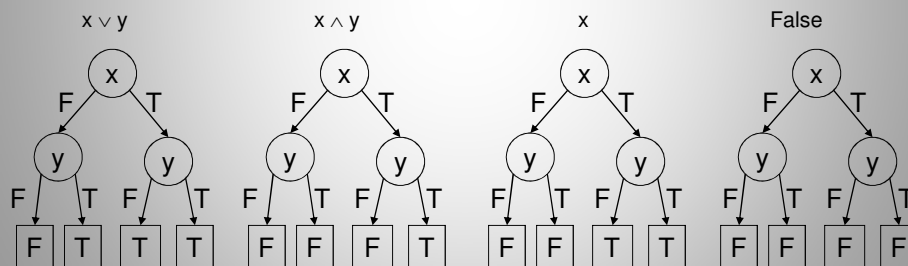
- We are interested in **Reduced** and **Ordered** Binary Decision Diagrams
- Reduced:
  - Merge all identical sub-trees in the binary decision tree (converts it to a directed-acyclic graph)
  - Remove redundant tests (if the false and true branches for a node go to the same place, remove that node)
- Ordered
  - We pick a fix order for the Boolean variables:
    - $x_0 < x_1 < x_2 < \dots$
  - The nodes in the BDD are listed based on this ordering

## Binary Decision Trees

Fix a variable order.

In each level of the tree, branch on the value of the variable in that level.

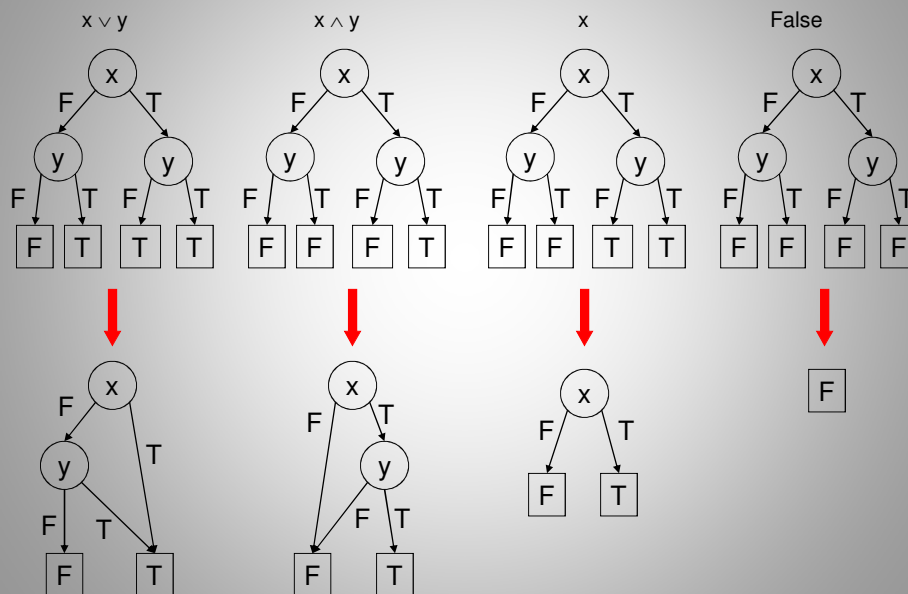
- Examples for boolean formulas on two variables  
Variable order:  $x, y$



## BDDs

- Repeatedly apply the following transformations to a binary decision tree:
  - Remove duplicate terminals
  - Remove duplicate non-terminals
  - Remove redundant tests
- These transformations transform the tree to a directed acyclic graph

## Binary Decision Trees vs. BDDs



## Good News About BDDs

- Given BDDs for two boolean logic formulas  $F$  and  $G$ 
  - The BDDs for  $F \wedge G$  and  $F \vee G$  are of size  $|F| \times |G|$  (and can be computed in that time)
  - The BDD for  $\neg F$  is of size  $|F|$  (and can be computed in that time)
  - $F \equiv G$  can be checked in linear time
  - Satisfiability of  $F$  can be checked in constant time
    - No, this does not mean that you can solve SAT in constant time

## Bad News About BDDs

- The size of a BDD can be exponential in the number of boolean variables
- The sizes of the BDDs are very sensitive to the variable ordering. Bad variable ordering can cause exponential increase in the size of the BDD
- There are functions which have BDDs that are exponential for any variable ordering (for example binary multiplication)
- Pre condition (EX) computation requires a sequence of existential variable eliminations
  - A sequence of existential variable eliminations can cause an exponential blow-up in the size of the BDD

## BDD Features

- Canonicity
  - If two functions are equal than their BDD graphs are isomorphic
- Compactness
  - Constant Boolean functions are one-node BDD
  - Some important functions grow linearly with number of input variables (e.g. **Parity**)
- Equivalence Check
  - Equivalence check is done in constant time

## BDD Features

- Basic Boolean Operations
  - The *size* of result of operation like  $F \wedge G$  is bounded by a product of sizes of  $F$  and  $G$
  - Using Dynamic Programming techniques, the *time* is almost linear with the size
- Complement
  - The size of  $\neg F$  is equal to size of  $F$
  - The time is linear with the size
  - Advanced techniques (*attributed edges*) reduce the time to constant

## BDD Features

- Find Satisfying Assignment
  - The time is linear with the size of a BDD
- Shared BDD
  - All features remain true for multiple boolean functions represented by BDD set

## Compare The Representations

	<b>Truth Table</b>	<b>Expression</b>	<b>Circuit</b>	<b>Decision Tree</b>	<b><i>BDD</i></b>
<b>Size</b>	Exponential	Linear	Linear	Exponential	<i>Linear to Exponential</i>
<b>Boolean Operator</b>	Linear	Constant	Constant	Linear	<i>Linear to Polynomial</i>
<b>Equality</b>	Linear	Exponential	Exponential	Linear	<i>Constant</i>
<b>Satisfy</b>	Linear	Exponential	Exponential	Linear	<i>Linear</i>
<b>Canonicity</b>	Yes	No	No	Yes	<i>Yes</i>

## Building a BDD

- Models Are Boolean Expressions/Circuits
- Naïve Approach
  - Build a decision tree and then apply reduction rules
  - Won't work: decision trees are too big
- Build BDD From Expressions
  - Use bottom-up strategy
  - Always start from "atomic" functions  $F = x$

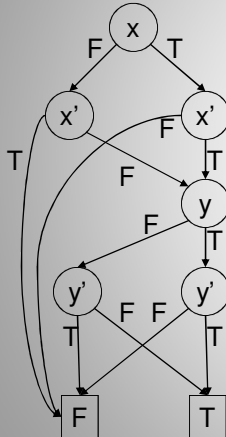
## Variables Ordering

- Good Order Is Crucial
  - Different variable orders may change the size of BDD from exponential to linear
- Theoretical Fact
  - Finding an optimal order is really hard
  - Best known algorithms have complexity  $O(3^N)$

## BDDs are Sensitive to Variable Ordering

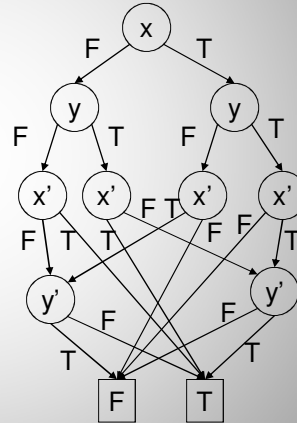
Identity relation for two variables:  $(x' \leftrightarrow x) \wedge (y' \leftrightarrow y)$

Variable order:  $x, x', y, y'$



For  $n$  variables,  $3n+2$  nodes

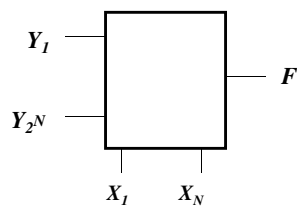
Variable order:  $x, y, x', y'$



For  $n$  variables,  $3 \times 2^n - 1$  nodes

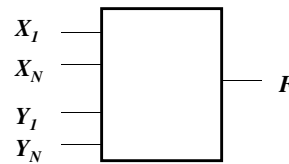
## Variables Ordering: Examples

### MUX



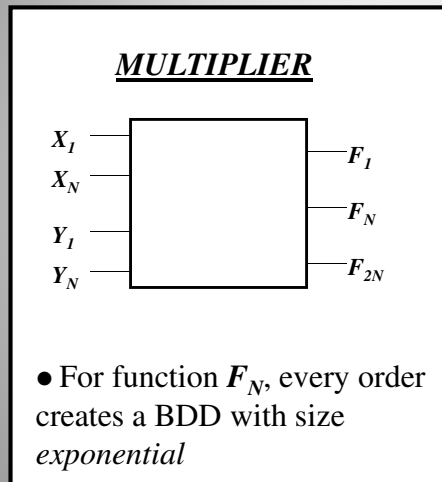
- If  $Y$ -variables are put first, the BDD size is *exponential*
- If  $X$ -variables are put first, the BDD size is *linear*

### COMPARATOR



- If  $X$ -variables are put before  $Y$ -variables, the size is *exponential*
- If order is interlaced ( $X_1, Y_1, \dots, X_N, Y_N$ ) the size is *linear*

## Variables Ordering: Examples

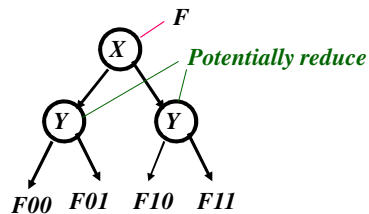


## Obtaining A Good Order

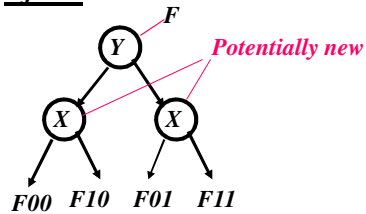
- Static Ordering
  - Basic idea: analyze the model's structure
- Dynamic Reordering
  - One basic idea: perform local order changes
    - Swap adjacent variables
    - Try a single variable at a time

## Basic Reorder Operation: Swap

**Before:**



**After:**



- Local Change
  - $X$  and  $Y$  are adjacent in the order
- Correctness Proof
  - Shannon expansion on both  $X$  and  $Y$  variables
- BDD Size Reduction
  - Each "atomic" swap removes *at most* 2 nodes
  - Swapping  $X$ - and  $Y$ -level nodes may reduce as much nodes as on  $Y$ -level

## Reordering Algorithm: Sifting

**SiftVariable (X)**

**WHILE**  $\neg$  *OnTop*(X)  
*Swap*(Prev(X),X); *SAVE Optimal*  
**WHILE**  $\neg$  *OnBottom*(X)  
*Swap*(X,Succ(X)); *SAVE Optimal*  
*MoveTo*(X, *Optimal*)

**Sifting(Variables)**

*Prioritize*(Variables)  
**FOREACH** X in Variables  
*SiftVariable*(X)

**ConvergenceSifting(Variables)**

**DO** *SiftingReorder*(Variables)  
**WHILE** *ReductionSize* > *Thresh*

- Performance
  - Complexity of **SiftVariable** is linear with the size of BDD
  - Complexity of **Sifting** is linear with  $N$  times the size of BDD
  - **ConvergenceSifting** is basically more powerful, but may be very expensive

## Reordering Variables: Advanced

- Selective Reordering
  - Reorder only variables which are 'important'
  - Reduces the reordering time
- Grouping Variables
  - Choose variables upon certain criterion (*symmetry, dependency*) and stick them together in groups
  - Reorder on the group level only
  - Better BDD size reduction in some cases