# Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams

Randal E. Bryant

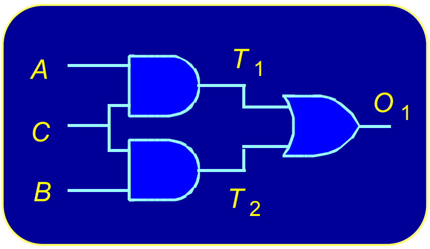
*Carnegie Mellon University*

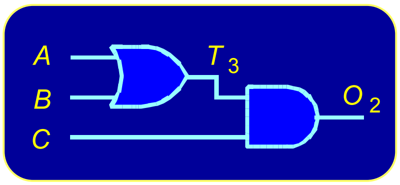
http://www.cs.cmu.edu/~bryant

## Example Analysis Task

### Logic Circuit Comparison

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





– 2 –

## Solution by Combinatorial Search

### Satisfiability Formulation

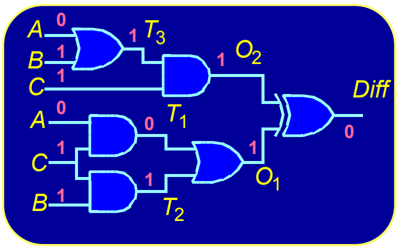
- Search for input assignment giving different outputs

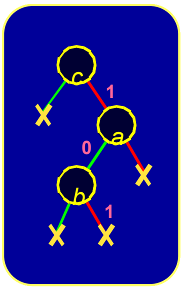
### Branch & Bound

- Assign input(s)
- Propagate forced values
- Backtrack when cannot succeed

### Challenge

- Must prove all assignments fail
  - Co-NP complete problem
- Typically explore significant fraction of inputs
- Exponential time complexity

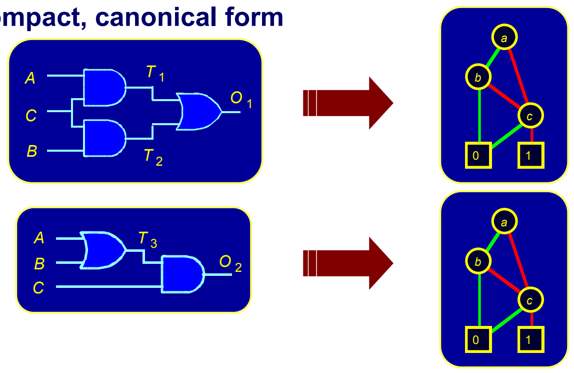




## Alternate Approach

### Generate Complete Representation of Circuit Function

- Compact, canonical form



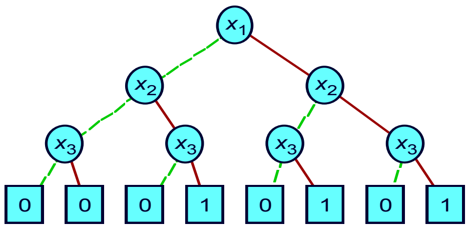
- Functions equal if and only if representations identical
- Never enumerate explicit function values
- Exploit structure & regularity of circuit functions

## Decision Structures

**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 |

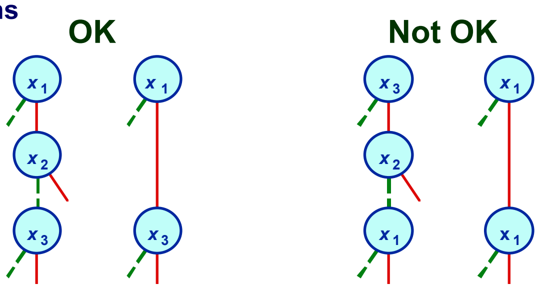
**Decision Tree**



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

## Variable Ordering

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

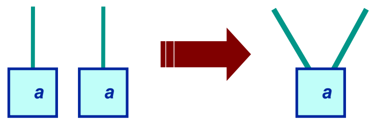


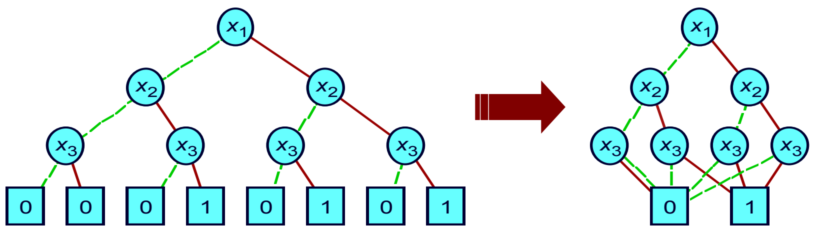
**Properties**

- No conflicting variable assignments along path
- Simplifies manipulation

## Reduction Rule #1

**Merge equivalent leaves**

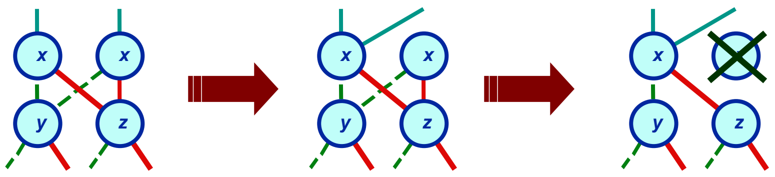


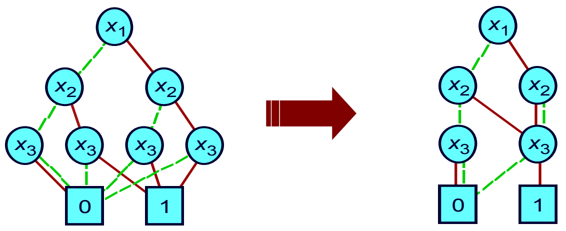


– 7 –

## Reduction Rule #2

**Merge isomorphic nodes**

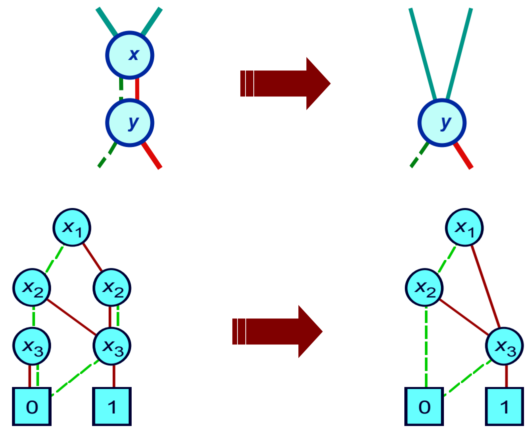




– 8 –

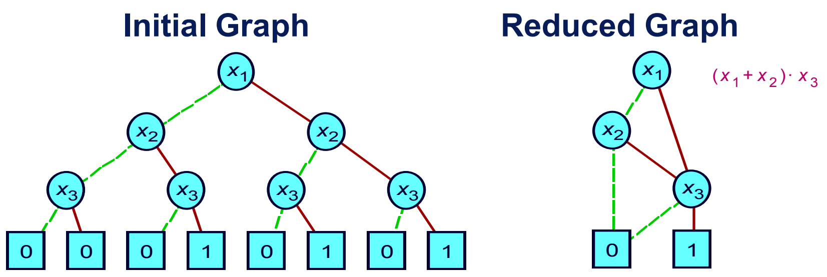
## Reduction Rule #3

### Eliminate Redundant Tests



– 9 –

## Example OBDD



### Canonical representation of Boolean function

- For given variable ordering
- Two functions equivalent if and only if graphs isomorphic
  - Can be tested in linear time
- Desirable property: *simplest form is canonical.*

– 10 –

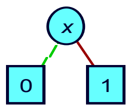
## Example Functions

### Constants

$\boxed{0}$ Unique unsatisfiable function

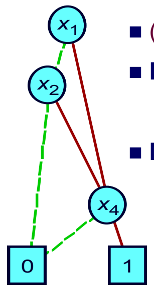
$\boxed{1}$ Unique tautology

### Variable



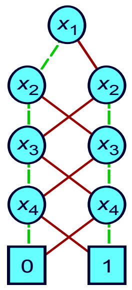
Treat variable as function

### Typical Function



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

### Odd Parity

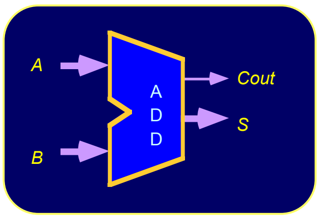


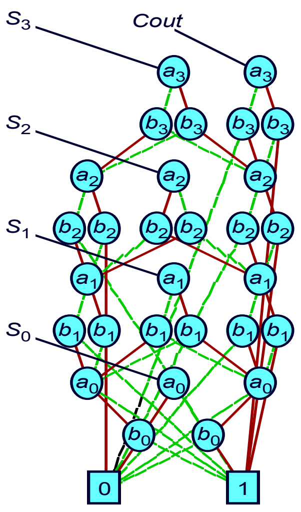
Linear representation

## Representing Circuit Functions

### Functions

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





### Shared Representation

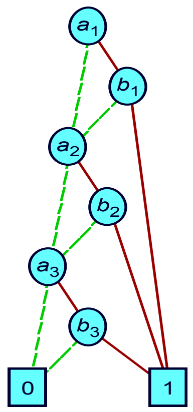
- Graph with multiple roots
- 31 nodes for 4-bit adder
- 571 nodes for 64-bit adder

☒ *Linear growth*

## Effect of Variable Ordering

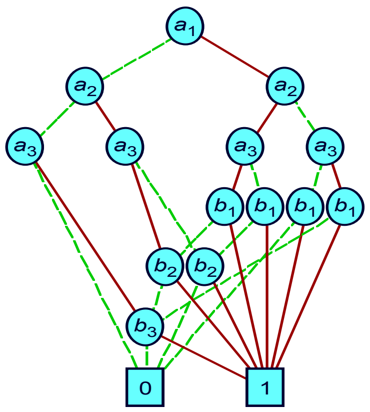
$(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$

**Good Ordering**



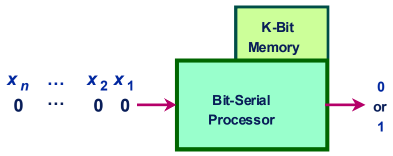
**Linear Growth**

**Bad Ordering**



**Exponential Growth**

## Bit Serial Computer Analogy



### Operation

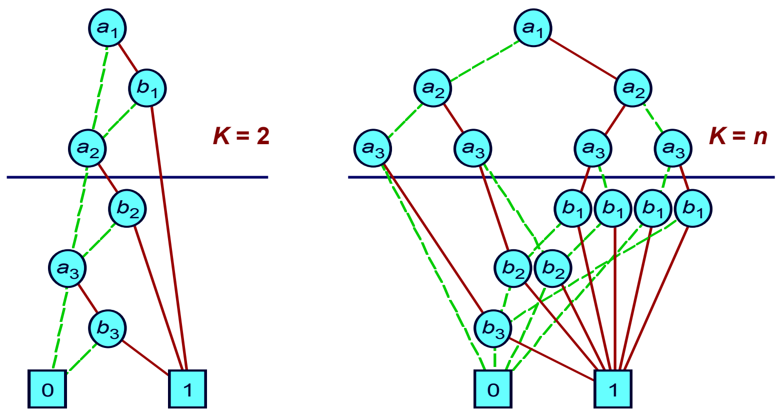
- Read inputs in sequence; produce 0 or 1 as function value.
- Store information about previous inputs to correctly deduce function value from remaining inputs.

### Relation to OBDD Size

- Processor requires $K$ bits of memory at step i.
- OBDD has ~ $2^K$ branches crossing level i.

## Analysis of Ordering Examples

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



## Selecting Good Variable Ordering

### Intractable Problem

- Even when problem represented as OBDD
  - I.e., to find optimum improvement to current ordering

### Application-Based Heuristics

- Exploit characteristics of application
- E.g., 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 Reordering

- Richard Rudell, Synopsys

### Periodically Attempt to Improve Ordering for All BDDs

- Part of garbage collection
- Move each variable through ordering to find its best location

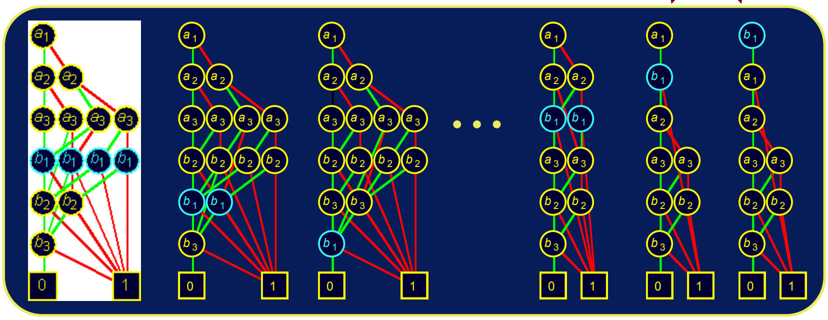
### Has Proved Very Successful

- Time consuming but effective
- Especially for sequential circuit analysis

## Dynamic Reordering By Sifting

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

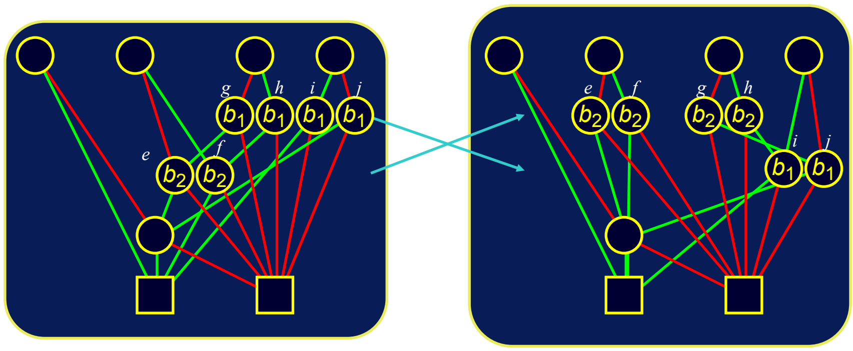
Best Choices



## Swapping Adjacent Variables

### Localized Effect

- Add / delete / alter only nodes labeled by swapping variables
- Do not change any incoming pointers



## 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

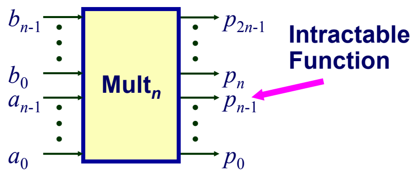
- Many tasks have reasonable OBDD representations
- Algorithms remain practical for up to 100,000 node OBDDs
- Heuristic ordering methods generally satisfactory

## Lower Bound for Multiplication

- Bryant, 1991

### Integer Multiplier Circuit

- *n*-bit input words *A* and *B*
- 2*n*-bit output word *P*



### Boolean function

- Middle bit (*n*-1) of product

### Complexity

- Exponential OBDD for all possible variable orderings

### Actual Numbers

- 40,563,945 BDD nodes to represent all outputs of 16-bit multiplier
- Grows 2.86x per bit of word size

## Symbolic Manipulation with OBDDs

### 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”

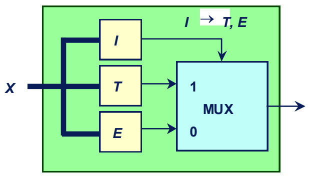
### Contrast to Traditional Approaches

- Apply search algorithm directly to problem representation
  - E.g., search for satisfying truth assignment to Boolean expression.

## If-Then-Else Operation

### Concept

- Basic technique for building OBDD from logic network or formula.



### Arguments *I*, *T*, *E*

- Functions over variables *X*
- Represented as OBDDs

### Result

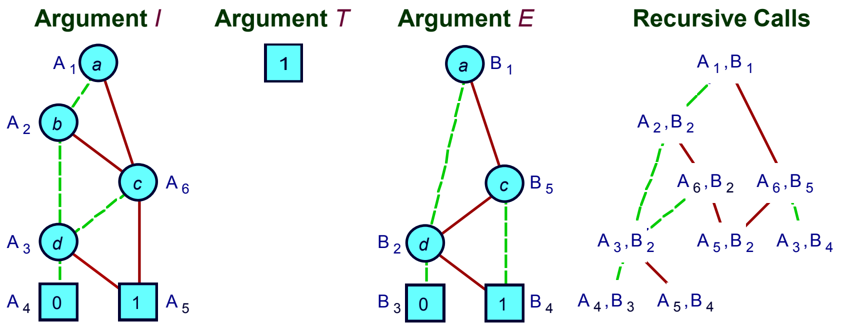
- OBDD representing composite function
- $(I \wedge T) \vee (\neg I \wedge E)$

### Implementation

- Combination of depth-first traversal and dynamic programming.
- Worst case complexity product of argument graph sizes.

– 23 –

## If-Then-Else Execution Example

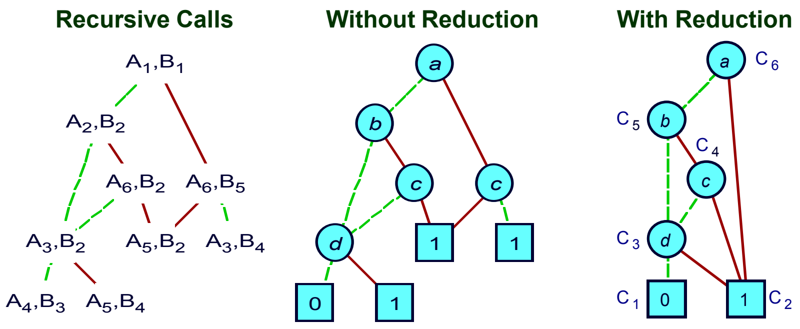


### Optimizations

- Dynamic programming
- Early termination rules

– 24 –

## If-Then-Else Result Generation



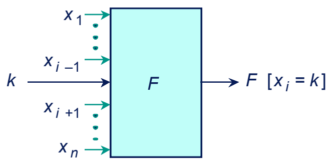
- Recursive calling structure implicitly defines unreduced BDD
- Apply reduction rules bottom-up as return from recursive calls
  - Generates reduced graph

## Restriction Operation

### Concept

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

| | | |
|---|---|---|
| $F_x$ | equivalent to | $F\,[x = 1]$ |
| $F_{\overline{x}}$ | equivalent to | $F\,[x = 0]$ |



### Implementation

- Depth-first traversal.
- Complexity near-linear in argument graph size

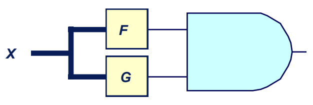
## Derived Operations

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

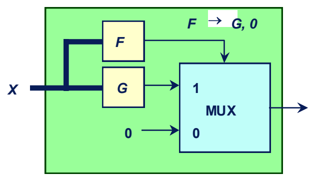
## Derived Algebraic Operations

- **Other operations can be expressed in terms of If-Then-Else**

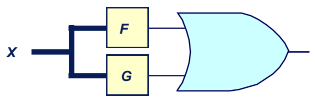
And(*F*, *G*)



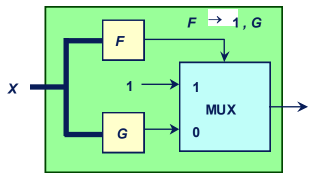
If-Then-Else(*F*, *G*, 0)



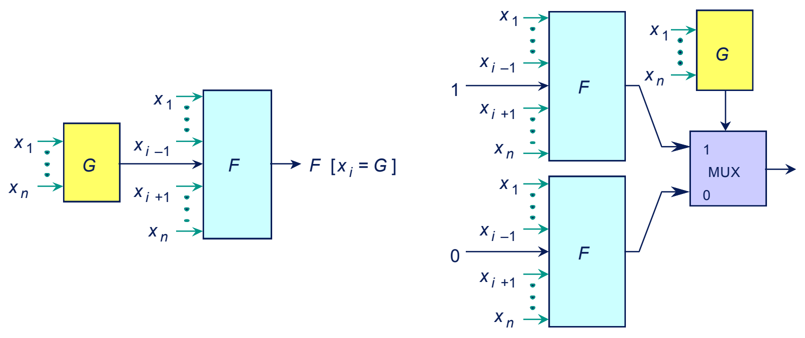
Or(*F*, *G*)



If-Then-Else(*F*, 1, *G*)

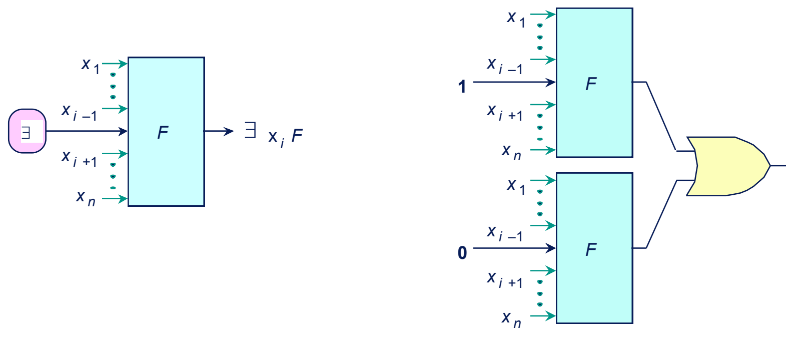


## Functional Composition



- Create new function by composing functions F and G.
- Useful for composing hierarchical modules.

## 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, Siemens, 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)

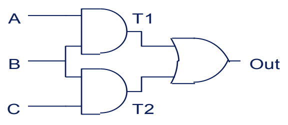
### Test

- False path identification (TI)

## Generating OBDD from Network

**Task:** Represent output functions of gate network as OBDDs.

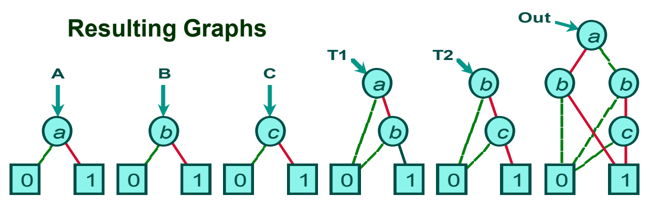
**Network**



**Evaluation**

```
A       ew_var ("a");
B       ew_var ("b");
C       ew_var ("c");
T1      And (A, 0, B);
T2      nd (B, C);
Out     r (T1, T2);
```

**Resulting Graphs**

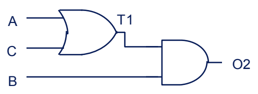


## Checking Network Equivalence

**Task: Do two networks compute same Boolean function?**

**Method: Compute OBDDs for both networks and compare**

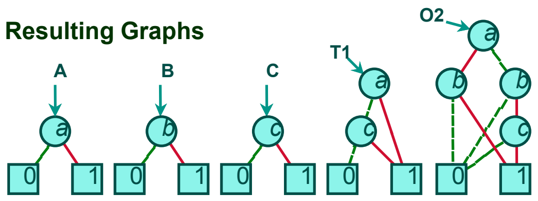
**Alternate Network**



**Evaluation**

```
T1    Or (A, C);
O2    And (T1, B);
if (O2 == Out)
      then Equivalent
      else Different
```

**Resulting Graphs**



– 33 –

## Finite State System Analysis

**Systems Represented as Finite State Machines**

- **Sequential circuits**
- **Communication protocols**
- **Synchronization programs**

**Analysis Tasks**

- **State reachability**
- **State machine comparison**
- **Temporal logic model checking**

**Traditional Methods Impractical for Large Machines**

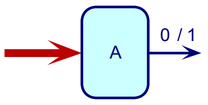
- **Polynomial in number of states**
- **Number of states exponential in number of state variables.**
- **Example: single 32-bit register has 4,294,967,296 states!**

– 34 –

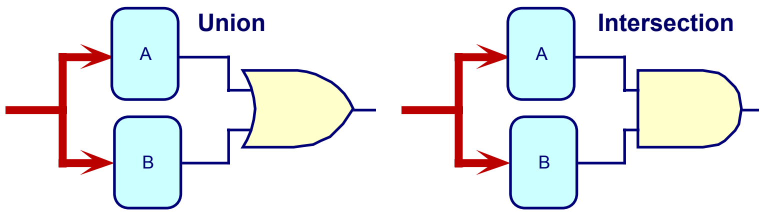
## Characteristic Functions

### Concept

- $A \subseteq \{0,1\}^n$
  - Set of bit vectors of length *n*
- **Represent set *A* as Boolean function A of *n* variables**
  - $X \in A$ if and only if $A(X) = 1$

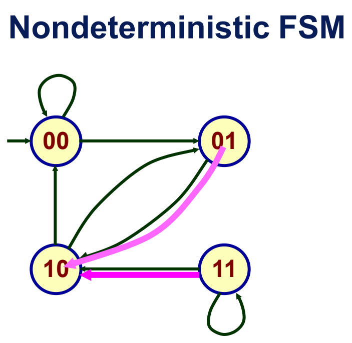


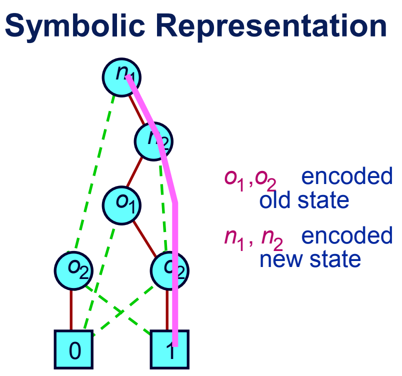
### Set Operations



– 35 –

## Symbolic FSM Representation





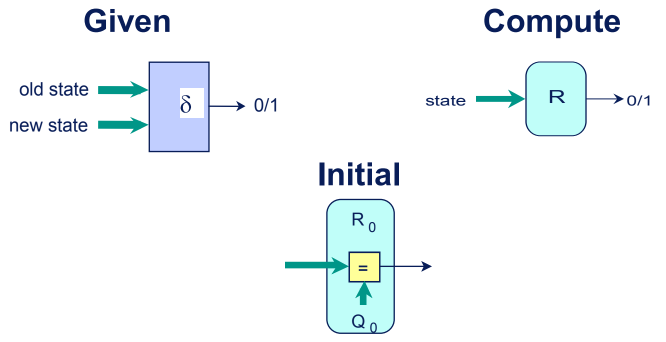
- **Represent set of transitions as function** $\delta$(*Old*, *New*)
  - Yields 1 if can have transition from state *Old* to state *New*
- **Represent as Boolean function**
  - Over variables encoding states

– 36 –

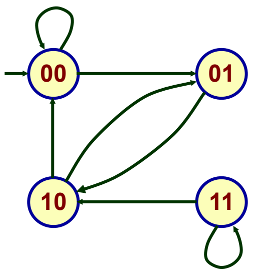
## Reachability Analysis

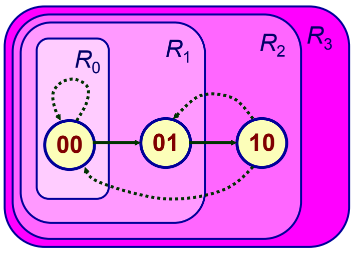
**Task**

- **Compute set of states reachable from initial state** $Q_0$
- **Represent as Boolean function** $R(S)$
- **Never enumerate states explicitly**



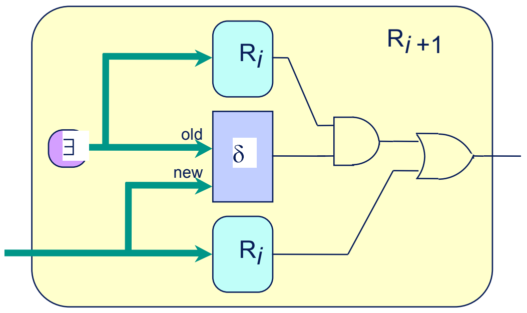
## Breadth-First Reachability Analysis





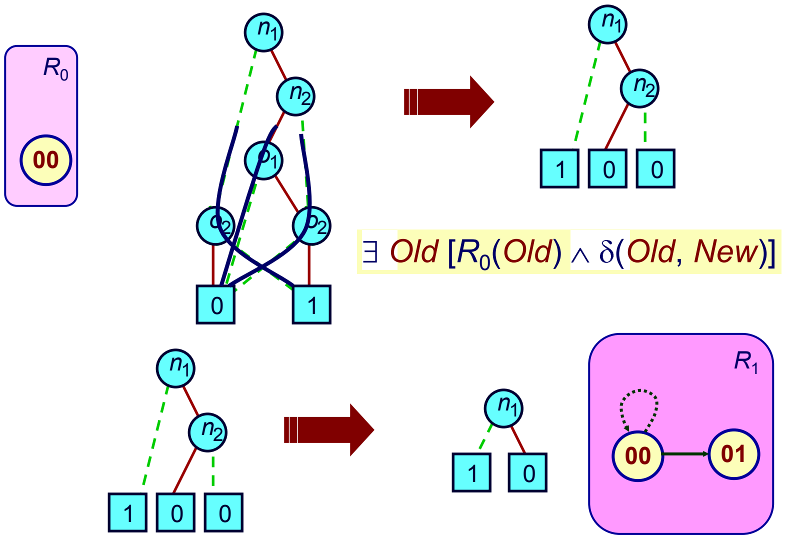
- $R_i$ **– set of states that can be reached in** $i$ **transitions**
- **Reach fixed point when** $R_n = R_{n+1}$
  - Guaranteed since finite state

## Iterative Computation



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

## Example: Computing $R_1$ from $R_0$



## Symbolic FSM Analysis Example

- K. McMillan, E. Clarke (CMU)   J. Schwalbe (Encore Computer)

### Encore Gigamax Cache System

- Distributed memory multiprocessor
- Cache system to improve access time
- Complex hardware and synchronization protocol.

### Verification

- Create “simplified” finite state model of system ($10^9$ states!)
- Verify properties about set of reachable states

### Bug Detected

- Sequence of 13 bus events leading to deadlock
- With random simulations, would require ≈2 years to generate failing case.
- In real system, would yield MTBF < 1 day.

## 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

- No other method comes close in overall strength
- Especially with quantification operations

## 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**
  - Critical effect on efficiency
  - Must have insights into problem characteristics
  - Dynamic reordering most promising workaround
- **Some operations too hard**
  - Must work around limitations

– 43 –

## Relaxing Ordering Requirement

### Challenge

- **Ordering is key to important properties of OBDDs**
  - Canonical form
  - Efficient algorithms for operating on functions
- **Some classes of functions have no good BDD orderings**
  - Graphs grow exponentially in all cases
- **Would like to relax requirement**
  - but still preserve (most of) the algorithmic properties

### Free Ordering

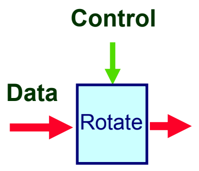
- **Gergov & Meinel, Sieling & Wegener**
- **Slight relaxation of ordering requirement**

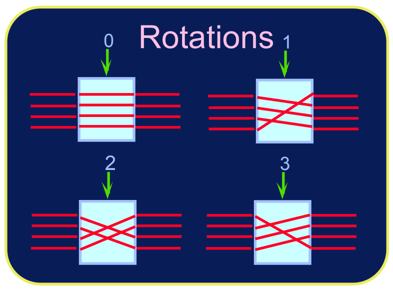
– 44 –

## Intractable OBDD Function Example

**Rotator**

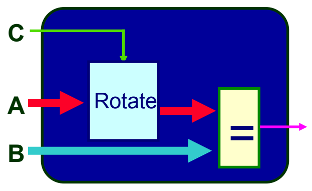
- Circular shift of data
- Shift amount set by control



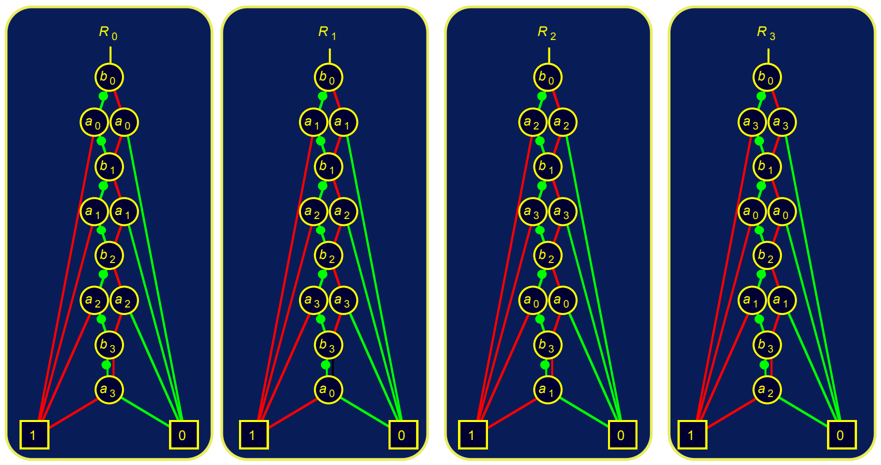


**Difficult Function**

- Rotate & compare

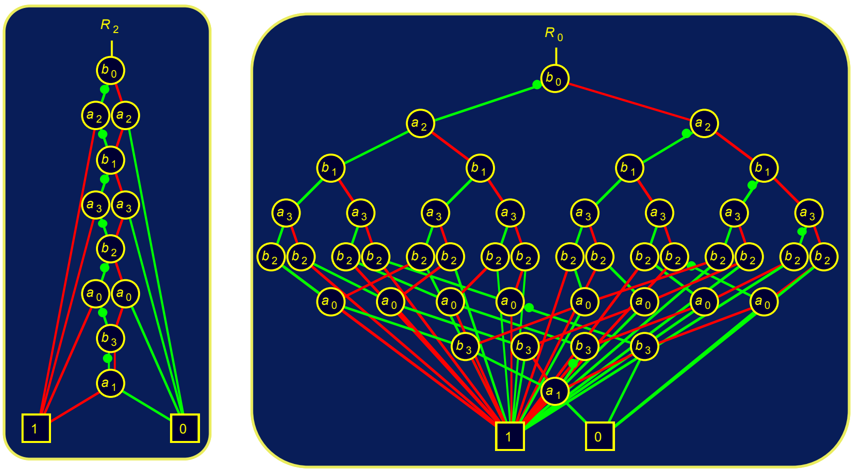


## OBDDs for Specific Rotations



- Can choose good ordering for any fixed rotation

## Forcing Single Ordering

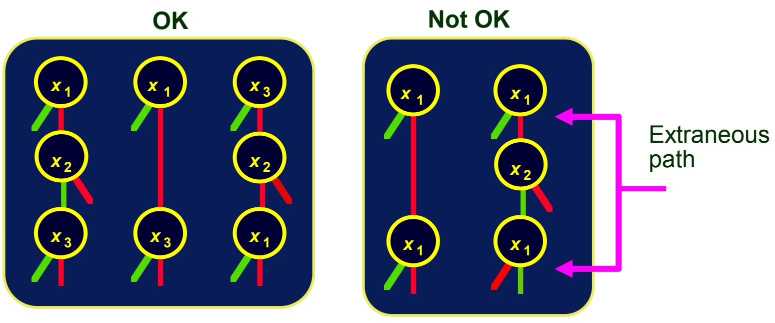


- Good ordering for one rotation terrible for another
- For any ordering, some rotation will have exponential OBDD

## Free BDDs

### Rules

- Variables may appear in any order
- Only allowed to test variable once along any path



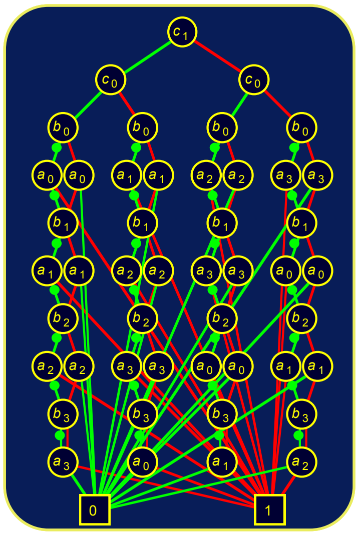
## Rotation Function Example

### Advantage

- Can select separate ordering for each rotation
- Good when different settings of control call for different orderings of data variables

### Still Has Limitations

- Representing output functions of multiplier
  - Exponential for all possible Free BDDs
  - Ponzio, '95



– 49 –

## Making Free BDDs Canonical

### Modified Ordering Requirement

- For any given variable assignment, variables must occur in fixed order
- But can vary from one assignment to another

### Algorithmic Properties Similar to OBDDs

- Reduce to canonical form
- Apply Boolean operation to functions
- Test for equivalence, satisfiability, etc.

### Some Operations Harder

- Variable quantification and composition
- But can restrict relevant variables to be totally ordered

– 50 –

## Representing Free Ordering

### Ordering Graph

- Encodes assignment-dependent variable ordering

### Similar to BDD

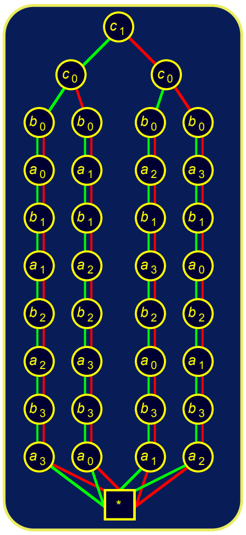
- Follow path according to assignment

### OBDD is Special Case

- Linear chain

### Ordering Requirement

- All functions must be compatible with single ordering graph



– 51 –

## Practical Aspects of Free BDDs

### Make Sense in Some Application Domain

- Usage of bits varies with context
- E.g., instruction set encodings

### Must Determine Good Ordering Graph

- Some success with heuristic methods
- Ideally should be done dynamically
  - Overwhelming degrees of freedom

### Need to Demonstrate Utility on Real-Life Examples

– 52 –