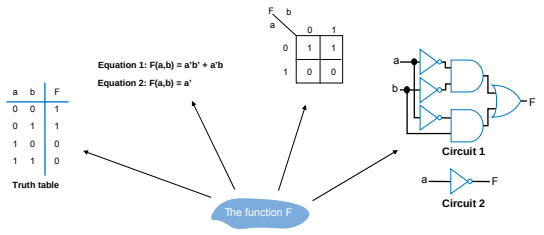
ECE 474A/57A
Computer-Aided Logic Design

Lecture 11
Binary Decision Diagrams (BDDs)

Boolean Logic Functions Representations

- Function can be represented in different ways
  - Truth table, equation, K-map, circuit, etc…
  - Some representations not unique (not canonical)

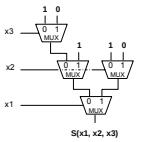


Why BDDs
An Efficient Representation

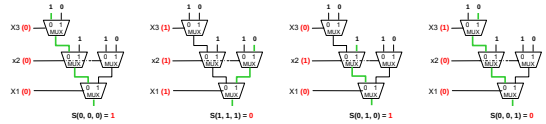
- Synthesis, optimization, verification, and testing algorithms/tools manipulate large Boolean functions
  - Important to have efficient way to represent these functions
  - Binary Decision Diagrams (BDDs) have emerged as a popular choice for representing these functions
- BDDs
  - Graph representation similar to a binary tree (i.e. decision trees from previous lectures)
  - Able to efficiently represent large functions
  - Some representations are canonical (unique)

Mux Representation of Boolean Functions

- MUX circuit to implement logic function S



x1	x2	x3	S
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

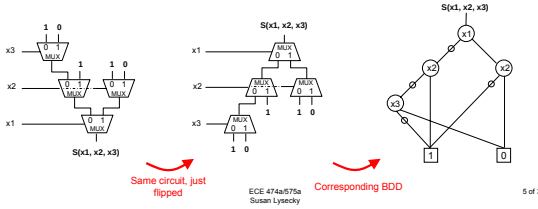


Mux Representation of Boolean Functions

Relation to BDDs

- Corresponding BDD to implement function S
  - One-to-one correspondence to the MUX gates in the flipped circuit

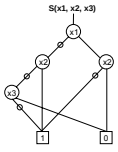
x1	x2	x3	S
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



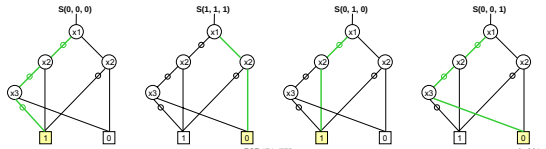
Binary Decision Diagram (BDD)

Example 1

- How does it work?
  - Line with bubble represent value = 0
  - Lines without bubble represent value = 1



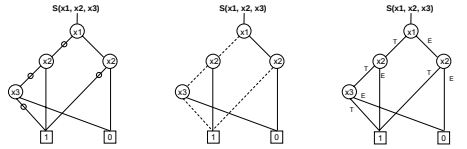
x1	x2	x3	S
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



Binary Decision Diagram (BDD)

Edge Notations

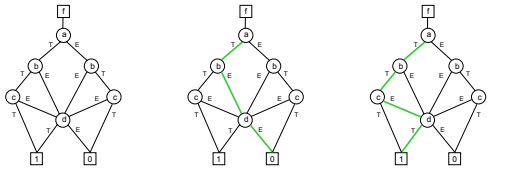
- Several ways to represent value = 1 and value = 0
  - Bubble vs. Non-bubble line
  - Dashed vs. Solid line
  - T (then) vs. E (else) labels
- We will adopt T vs. E labels – consistent with most of the book (Hatchel) examples



Binary Decision Diagram (BDD)

Example 2

- Let's consider another function
  f(a,b,c,d) = abc + b'd + c'd



What is the value of f(1,0,1,0)?

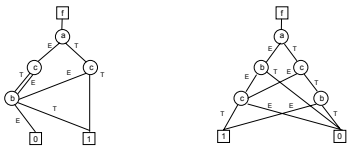
What is the value of f(1,1,0,1)?

Notice that if a=1 and b=0, the function does not depend on a value for c.

Ordered Binary Decision Diagram (OBDD)

What is a OBDD?

- Ordered binary decision diagrams ensure the variables appear in the same order along all paths from the root to the leaves



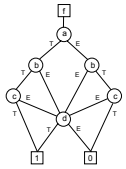
Ordering : a ≤ c ≤ b

Not ordered

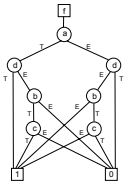
Ordered Binary Decision Diagram (OBDD)

Different Ordering Lead to Different Complexity – Example1

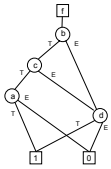
- Variable ordering important, may result in a more complex (or simple) BDD
  - All three BDDs below represent the same function
  - Third ordering (b ≤ c ≤ a ≤ d) optimal because there is exactly one node for each variable



Order : a ≤ b ≤ c ≤ d



Order : a ≤ d ≤ b ≤ c

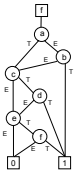


Order : b ≤ c ≤ a ≤ d

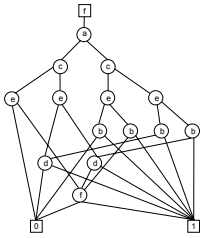
Ordered Binary Decision Diagram (OBDD)

Different Ordering Lead to Different Complexity – Example 2

- Consider F = ab + cd + ef, again both BDDs represent same function
- Variable order has a large impact on resulting BDD, first variable ordering (a ≤ b ≤ c ≤ d ≤ e ≤ f) yields a much simpler BDD



Order : a ≤ b ≤ c ≤ d ≤ e ≤ f



Order : a ≤ c ≤ e ≤ b ≤ d ≤ f

BDDs for Basic Logic Functions

NOT

a	F
0	1
1	0

AND

a	b	F
0	0	0
0	1	0
1	0	0
1	1	1

OR

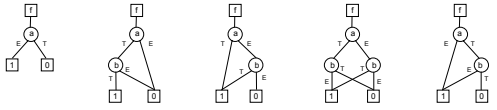
a	b	F
0	0	0
0	1	1
1	0	1
1	1	1

XOR

a	b	F
0	0	0
0	1	1
1	0	1
1	1	0

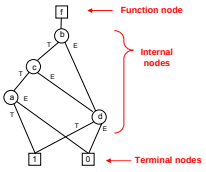
NAND

a	b	F
0	0	1
0	1	1
1	0	1
1	1	0



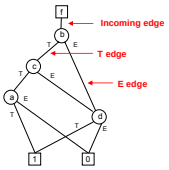
Formal Definition of BDDs

- A BDD is a direct acyclic graph (DAG) representing a multiple-output switching function F
  - Nodes are partitioned into three subsets
    - Function node
      - Represents the function symbol (f)
      - Indegree = 0
      - Outdegree = 1
    - Internal node
      - Represents variable in function (a, b, c, d)
      - Indegree ≥ 1
      - Outdegree = 2
    - Terminal node
      - Represents a value (1 or 0)
      - Indegree ≥ 1
      - Outdegree = 0



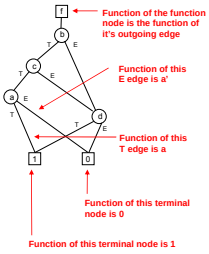
Formal Definition of BDDs

- A BDD definition (cont')
  - Three types of edges
    - Incoming edge
      - From the function node and defines function
    - T edge
      - From an internal node and represents when the corresponding variable is 1
    - E edge
      - From an internal node and represents when the corresponding variable is 0



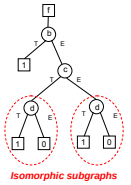
Formal Definition of BDDs

- A BDD definition (cont')
  - The function f represented by a BDD is defined as follows
    - The function of the terminal node is a constant value (1 or 0)
    - The function of a T edge is the function of the head node
    - The function of a E edge is the complement of the function of the node
    - The function of a node v is given by $vf_T + v'f_{E'}$ when f_T is the function of the T edge and f_E is the function of the E edge
    - The function of the function node is the function of it's outgoing edge



BDD Canonical Form

- BDDs are canonical (unique) for a representation of F given a variable ordering if
  - All internal nodes are descendants of some node
  - There are no isomorphic subgraphs
  - For every node $f_T \neq f_E$



Isomorphic subgraphs

Isomorphic
Two graphs are isomorphic if there is a one-to-one correspondence between their vertices and there is an edge between two vertices of one graph if and only if there is an edge between the two corresponding vertices in the other graph

English – same subgraph - all vertices the same, all edges between vertices the same

Building BDDs For a Function F

- How do I build a BDD given a function F ?
- Recursive use of Shannon's Expansion Theorem
  - $F = aF_a + a'F_{a'}$
  - We can keep applying expansion theorem, eventually we reach the unique canonical form , which uses only minterms

F = a'b + abc' + a'b'c
F = a(bc') + a'(b+b'c)

$F_a = (bc')$

$F_{a'} = (b+b'c)$

F_a also called the **cofactor** of F **w.r.t.** (with respect to) a

F = a'b + abc' + a'b'c
F = $bF_b + b'F_{b'}$
F = b(a'+ ac') + b'(a'c)

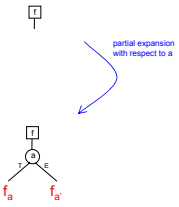
F expanded w.r.t to b

F = a'b + abc' + a'b'c
F = $cF_c + c'F_{c'}$
F = c(a'b + a'b') + c'(a'b + ab)

F expanded w.r.t to c

Building BDDs - Exercise 1

- Build a BDD for f = abc + ab'c + a'bc' + a'b'c'
- Use the variable ordering a ≤ b ≤ c



Compute cofactors of f with respect to a (first variable in ordering)

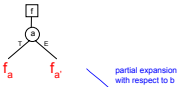
f = abc + ab'c + a'bc' + a'b'c'

f_a = bc + b'c

$f_{a'}$ = bc' + b'c'

Building BDDs - Exercise 1

- Build a BDD for f = abc + ab'c + a'bc' + a'b'c'
- Use the variable ordering a ≤ b ≤ c



partial expansion with respect to b

Compute cofactors of f_a and $f_{a'}$ with respect to b (second variable in ordering)

$f_a = bc + b'c$

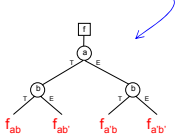
$(f_a)_b = f_{ab} = c$

$(f_a)_{b'} = f_{ab'} = c$

$f_{a'} = bc' + b'c'$

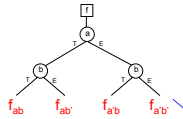
$(f_{a'})_b = f_{a'b} = c'$

$(f_{a'})_{b'} = f_{a'b'} = c'$



Building BDDs - Exercise 1

- Build a BDD for f = abc + ab'c + a'bc' + a'b'c'
- Use the variable ordering a ≤ b ≤ c



expansion with respect to c, final BDD

Compute cofactors of f_{ab}, $f_{ab'}$, $f_{a'b}$, $f_{a'b'}$ with respect to c (third variable in ordering)

$f_{ab} = c$

$f_{abc} = 1$

$f_{abc'} = 0$

$f_{ab'} = c$

$f_{ab'c} = 1$

$f_{ab'c'} = 0$

$f_{a'b} = c'$

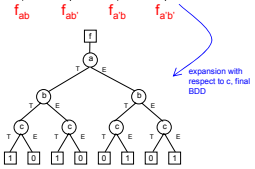
$f_{a'bc} = 0$

$f_{a'bc'} = 1$

$f_{a'b'} = c'$

$f_{a'b'c} = 0$

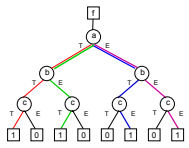
$f_{a'b'c'} = 1$



Building BDDs - Exercise 1

- f = abc + ab'c + a'bc' + a'b'c'

a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



Does it work?

Building BDDs - Exercise 2

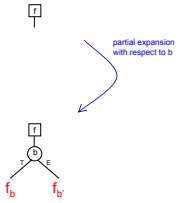
- Build a BDD for f = abc + b'd + c'd
- Use the variable ordering b ≤ c ≤ d ≤ a

Compute cofactors of f with respect to b (first variable in ordering)

f = abc + b'd + c'd

f_b = ac + c'd

$f_{b'}$ = d + c'd



Building BDDs - Exercise 2

- Build a BDD for f = abc + b'd + c'd
- Use the variable ordering b ≤ c ≤ d ≤ a

Compute cofactors of f_b and $f_{b'}$ with respect to c (second variable in ordering)

f_b = ac + c'd

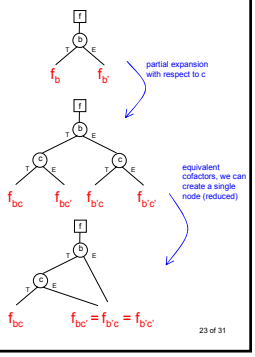
f_{bc} = a

$f_{bc'}$ = d

$f_{b'}$ = d + c'd

$f_{b'c}$ = d

$f_{b'c'}$ = d



Building BDDs - Exercise 2

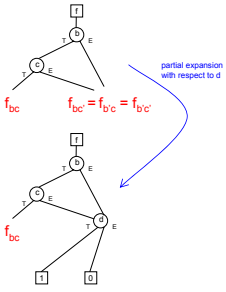
- Build a BDD for f = abc + b'd + c'd
- Use the variable ordering b ≤ c ≤ d ≤ a

Compute cofactors of f_{bc}, $f_{bc'}$ and $f_{b'c'}$ with respect to d (third variable in ordering)

$f_{bc'} = f_{bc'} = f_{b'c'}$ = d

$f_{bc'd}$ = $f_{bc'd}$ = $f_{b'c'd}$ = 1

$f_{bc'd'}$ = $f_{bc'd'}$ = $f_{b'c'd}$ = 0



Building BDDs - Exercise 2

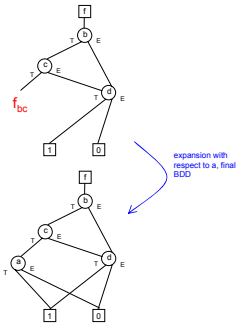
- Build a BDD for f = abc + b'd + c'd
- Use the variable ordering b ≤ c ≤ d ≤ a

Compute cofactors of f_{bc} with respect to a (fourth variable in ordering)

$f_{bc} = a$

$f_{bca} = 1$

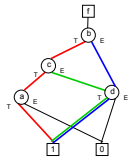
$f_{bca'} = 0$



Building BDDs - Exercise 2

- Build a BDD for f = abc + b'd + c'd

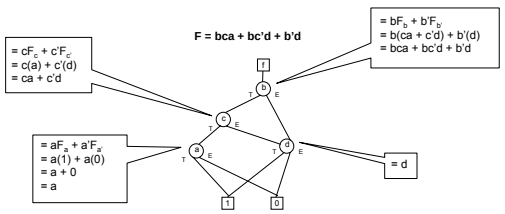
a	b	c	d	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



Does it work?

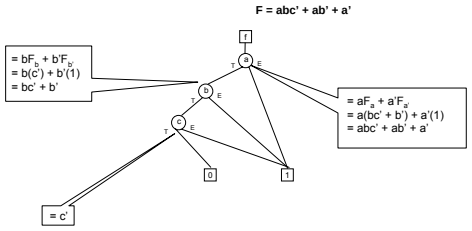
BDD to Boolean Function – Exercise 1

- Can we go from a BDD to a Boolean function ?
  - Sum all paths from function node to terminal nodes



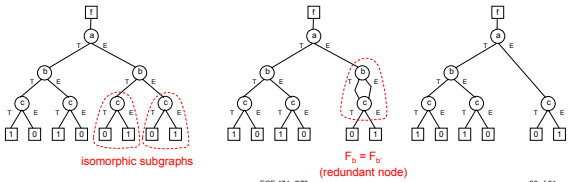
BDD to Boolean Function – Exercise 2

- Another Example



Reducing BDDs

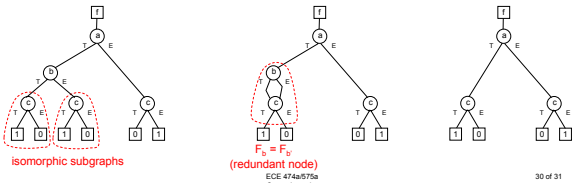
- When building BDDs, result not always reduced (Example 1 - slide 19)
  - We have isomorphic subgraphs, potential for reduction
- Transform a non-reduced BDD into a reduced BDD by iteratively applying
  - Identify isomorphic subgraphs
  - Remove redundant nodes
- A ordered reduced BDD (ROBDD) is unique so we have a canonical form



Reducing BDDs

Example 2

- Let's try to reduce and OBDD
  - Iterative apply
    - Identify isomorphic subgraphs
    - Remove redundant nodes



Summary, and then some…

- Binary Decision Diagrams (BDDs)
  - Efficient mechanism to representation of Boolean functions in terms of memory and CPU
  - Translating function→BDD and BDD→function
  - Importance of variable ordering
  - Method to reduced OBDDs, getting a ROBDD
- We said that BDDs can be efficiently stored and manipulated – HOW?
  - *Refer to handwritten notes*
  - *Supplemental materials attached*

ITE Operator

Two argument operators expressed in terms of ITE

Table	Name	Expression	Equivalent Form
0000	0	0	0
0001	AND(F, G)	FG	ITE(F, G, 0)
0010	F > G	FG′	ITE(F, G′, 0)
0011	F	F	F
0100	F < G	F′G	ITE(F, 0, G)
0101	G	G	G
0110	XOR(F, G)	F ⊕ G	ITE(F, G′, G)
0111	OR(F, G)	F + G	ITE(F, 1, G)
1000	NOR(F, G)	(F + G)′	ITE(F, 0, G′)
1001	XNOR(F, G)	(F ⊕ G)′	ITE(F, G, G′)
1010	NOT(G)	G′	ITE(G, 0, 1)
1011	F ≥ G	F + G′	ITE(F, 1, G′)
1100	NOT(F)	F′	ITE(F, 0, 1)
1101	F ≤ G	F′ + G	ITE(F, G, 1)
1110	NAND(F, G)	(FG)′	ITE(F, G′, 1)
1111	1	1	1

ITE Algorithm

- Most standard manipulation of BDDs can be done with ITE
- Algorithm recursive, based on formulation where v is the top variable of F, G, H

$$\begin{aligned} ITE(F, G, H) &= FG + F'H \\ &= v(FG + F'H)_v + v'(FG + F'H)_{v'} \\ &= v(F_vG_v + F'_vH_v) + v'(F_{v'}G_{v'} + F'_{v'}H_{v'}) \\ &= ITE(\, v, ITE(F_v, G_v, H_v), ITE(F_{v'}, G_{v'}, H_{v'}) \,) \end{aligned}$$

- Terminal cases are as follows

$$ITE(1, F, G) = ITE(0, G, F) = ITE(F, 1, 0) = ITE(G, F, F) = F$$

Pseduo-code of the ITE Algorithm

```
ITE(F, G, H){
    (result, terminal_case) = TERMINAL_CASE(F, G, H)          // did we find a terminal case?
    if (terminal_case) return (result)

    (result, in_computed_table) = COMPUTED_TABLE_HAS_ENTRY(F, G, H)   // have we already calculated this value?
    if (in_computed_table) return (result)

    v = TOP_VARIABLE(F, G, H)                                 // recursively calculate this value
    T = ITE(F_v, G_v, H_v)
    E = ITE(F_v', G_v', H_v')
    R = FIND_OR_ADD_UNIQUE_TABLE(v, T, E)                     // see if subtree already present
    INSERT_COMPUTED_TABLE((F, G, H), R)                       // record the calculated value
    return (R)
}
```