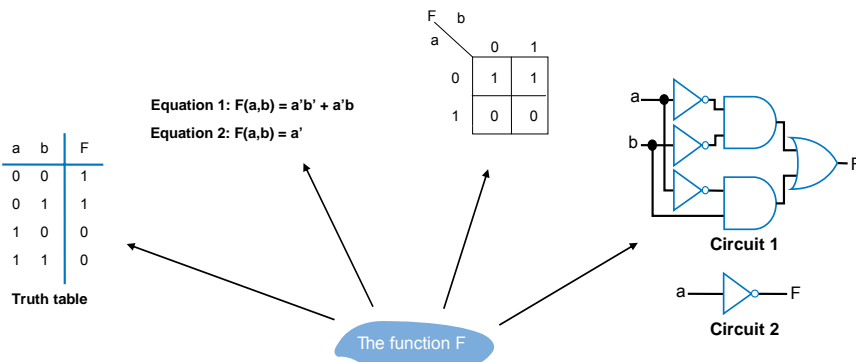


## Lecture 11 Binary Decision Diagrams (BDDs)

## Boolean Logic Functions Representations

---

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



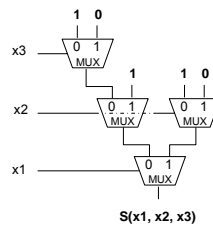
# Why BDDs

## An Efficient Representation

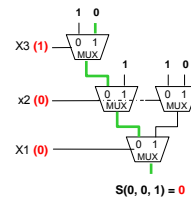
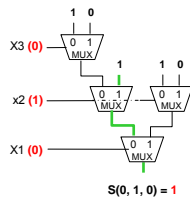
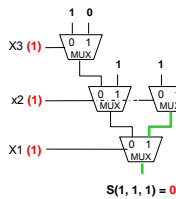
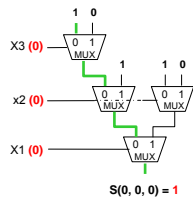
- Synthesis, optimization, verification, and testing algorithms/tools manipulate large Boolean functions
  - Important to have efficient way to represent these functions
  - Binary Decision Diagrams (BDDs) have emerged as a popular choice for representing these functions
- BDDs
  - Graph representation 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

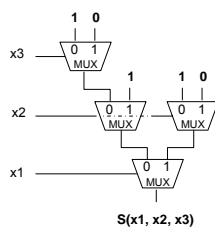


# Mux Representation of Boolean Functions

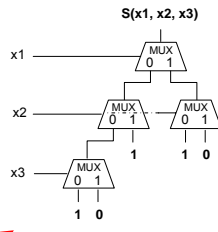
## Relation to BDDs

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

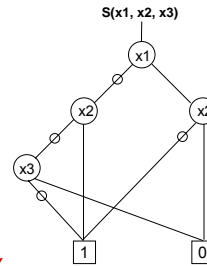
x1	x2	x3	S
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



Same circuit, just flipped



Corresponding BDD



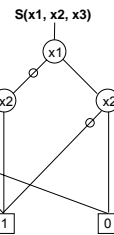
ECE 474a/575a  
Susan Lysecky

5 of 47

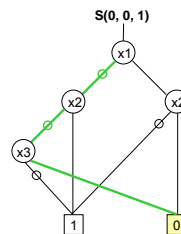
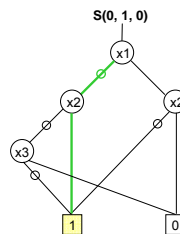
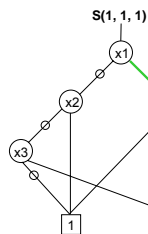
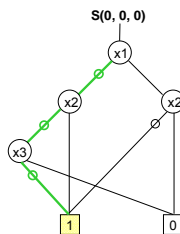
# Binary Decision Diagram (BDD)

## 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



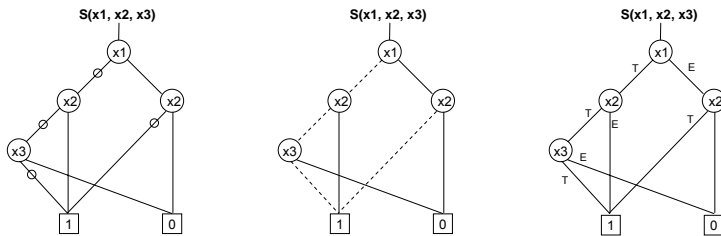
ECE 474a/575a  
Susan Lysecky

6 of 47

## 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



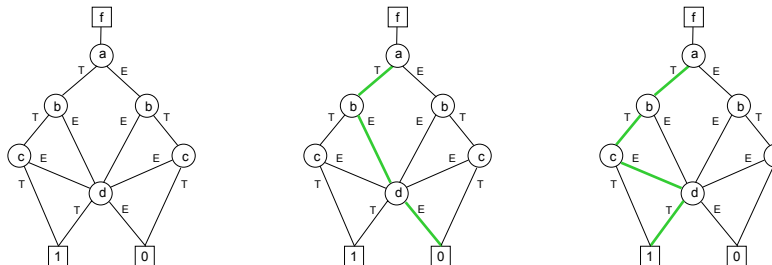
ECE 474a/575a  
Susan Lysecky

7 of 47

## 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  $c$ .

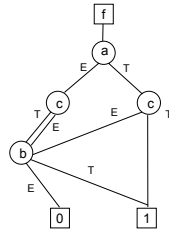
ECE 474a/575a  
Susan Lysecky

8 of 47

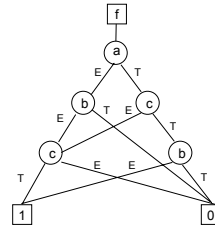
## Ordered Binary Decision Diagram (OBDD)

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 :  $a \leq c \leq b$



Not ordered

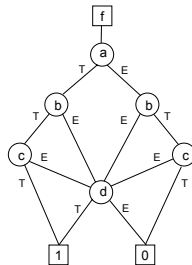
ECE 474a/575a  
Susan Lysecky

9 of 47

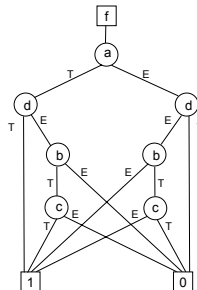
## Ordered Binary Decision Diagram (OBDD)

Different Ordering Lead to Different Complexity – Example 1

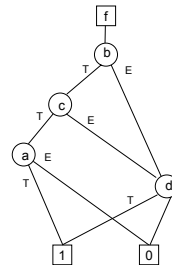
- Variable ordering important, may result in a more complex (or simple) BDD
  - All three BDDs below represent the same function
  - Third ordering ( $b \leq c \leq a \leq d$ ) optimal because there is exactly one node for each variable



Order :  $a \leq b \leq c \leq d$



Order :  $a \leq d \leq b \leq c$



Order :  $b \leq c \leq a \leq d$

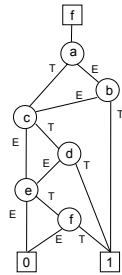
ECE 474a/575a  
Susan Lysecky

10 of 47

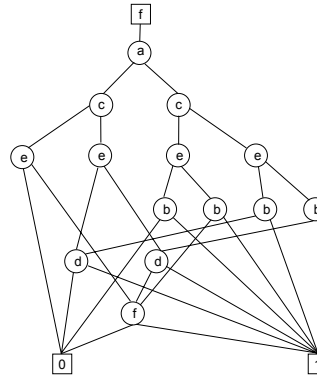
# Ordered Binary Decision Diagram (OBDD)

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 ( $a \leq b \leq c \leq d \leq e \leq f$ ) yields a much simpler BDD



Order :  $a \leq b \leq c \leq d \leq e \leq f$



Order :  $a \leq c \leq e \leq b \leq d \leq f$

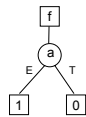
ECE 474a/575a  
Susan Lysecky

11 of 47

# BDDs for Basic Logic Functions

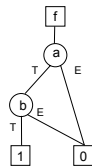
NOT

a	F
0	1
1	0



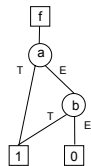
AND

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



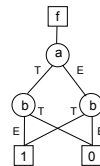
OR

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



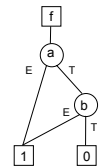
XOR

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



ECE 474a/575a  
Susan Lysecky

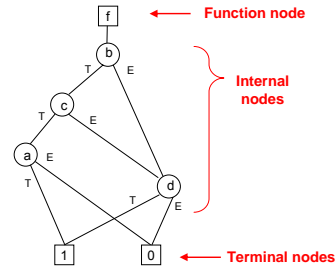
12 of 47

## Formal Definition of BDDs

- A BDD is a direct acyclic graph (DAG) representing a multiple-input switching function  $F$

- Nodes are partitioned into three subsets

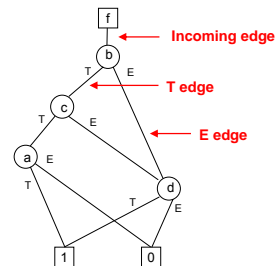
- Function node
  - Represents the function symbol ( $f$ )
  - Indegree = 0
  - Outdegree = 1
- Internal node
  - Represents variable in function ( $a, b, c, d$ )
  - Indegree  $\geq 1$
  - Outdegree = 2 (*actually we can optimize  $\leq 2$* )
- Terminal node
  - Represents a value (1 or 0)
  - Indegree  $\geq 1$
  - Outdegree = 0



## Formal Definition of BDDs

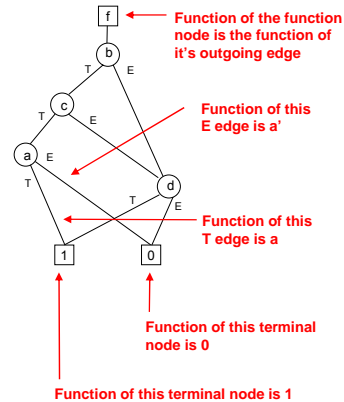
- A BDD definition (cont')

- 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



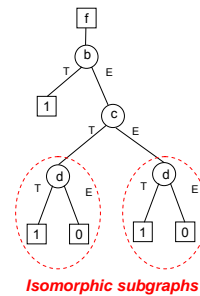
## Formal Definition of BDDs

- 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 an E edge is the complement of the function of the node
    - The function of a node  $v$  is given by  $v f_T + v' f_E$ , 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 its outgoing edge



## BDD Canonical Form

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



### 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

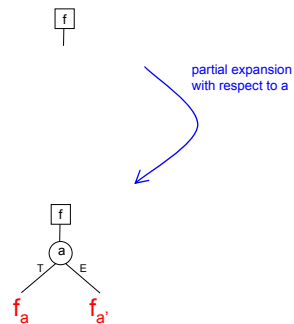
- Build a BDD for  $f = abc + ab'c + a'bc' + a'b'c'$
- Use the variable ordering  $a \leq b \leq c$

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  $a \leq b \leq c$

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

$$f_a = bc + b'c$$

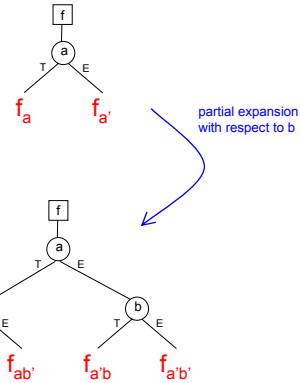
$$(f_a)_b = f_{ab} = c$$

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

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

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

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



## Building BDDs - Exercise 1

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

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

$$f_{ab} = c$$

$$f_{abc} = 1$$

$$f_{abc'} = 0$$

$$f_{ab'} = c$$

$$f_{ab'c} = 1$$

$$f_{ab'c'} = 0$$

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

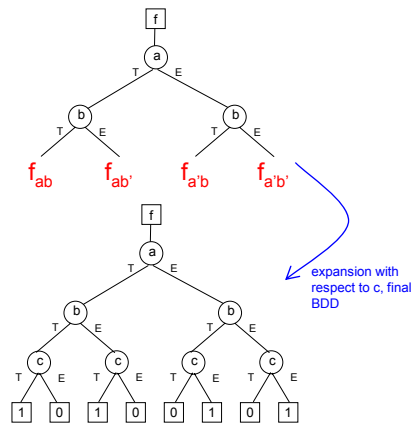
$$f_{a'bc} = 0$$

$$f_{a'bc'} = 1$$

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

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

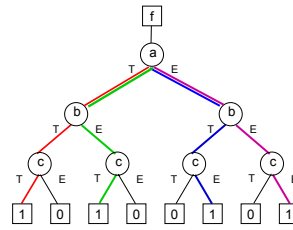
$$f_{a'b'c'} = 1$$



## 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

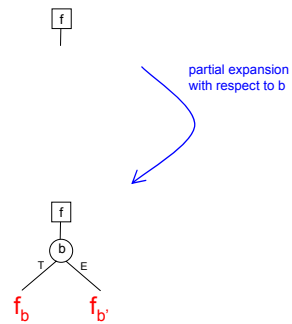
- Build a BDD for  $f = abc + b'd + c'd$
- Use the variable ordering  $b \leq c \leq d \leq a$

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  $b \leq c \leq d \leq a$

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

$$f_b = ac + c'd$$

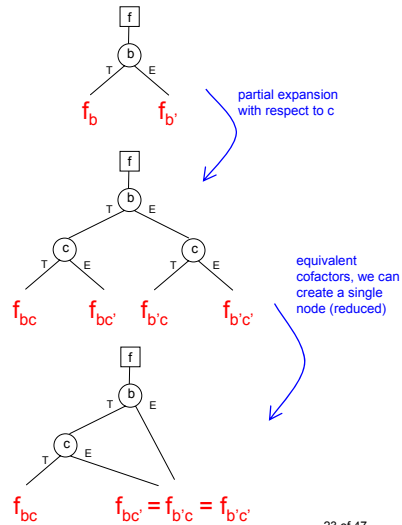
$$f_{bc} = a$$

$$f_{bc'} = d$$

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

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

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



ECE 474a/575a  
Susan Lysecky

23 of 47

## Building BDDs - Exercise 2

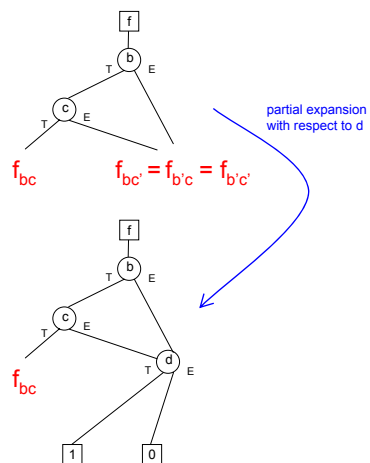
- Build a BDD for  $f = abc + b'd + c'd$
- Use the variable ordering  $b \leq c \leq d \leq a$

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

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

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

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



ECE 474a/575a  
Susan Lysecky

24 of 47

## Building BDDs - Exercise 2

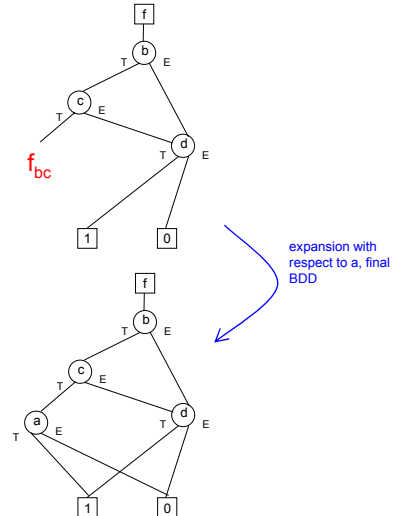
- Build a BDD for  $f = abc + b'd + c'd$
- Use the variable ordering  $b \leq c \leq d \leq a$

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

$$f_{bc} = a$$

$$f_{bca} = 1$$

$$f_{bca'} = 0$$



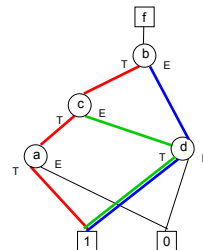
ECE 474a/575a  
Susan Lysecky

25 of 47

## 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



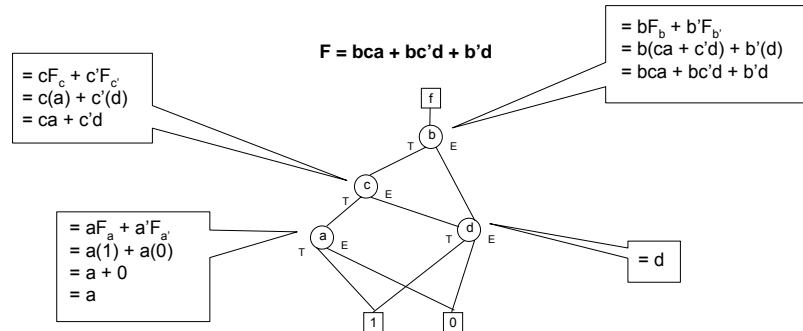
**Does it work?**

ECE 474a/575a  
Susan Lysecky

26 of 47

## 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

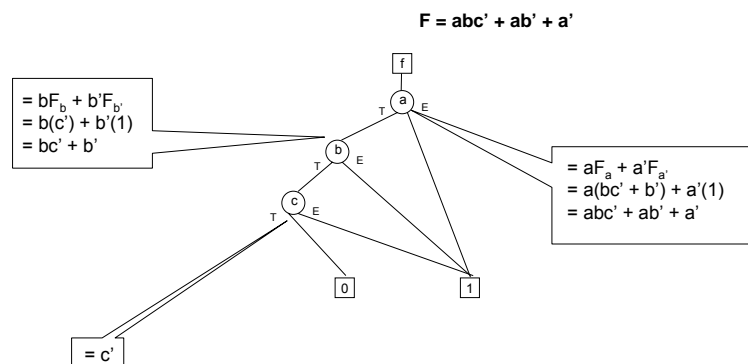


ECE 474a/575a  
Susan Lysecky

27 of 47

## BDD to Boolean Function – Exercise 2

- Another Example

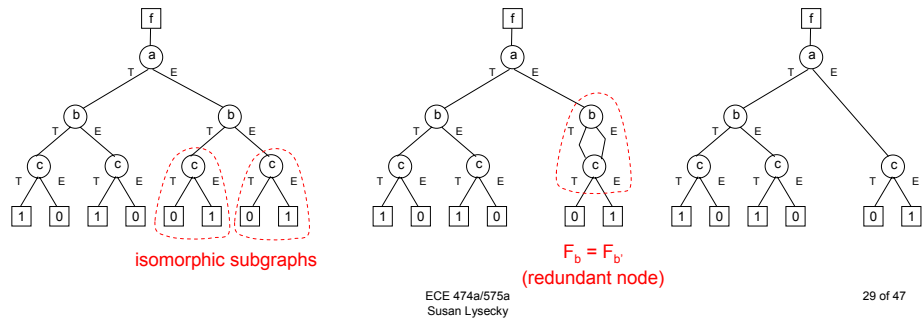


ECE 474a/575a  
Susan Lysecky

28 of 47

## Reducing BDDs

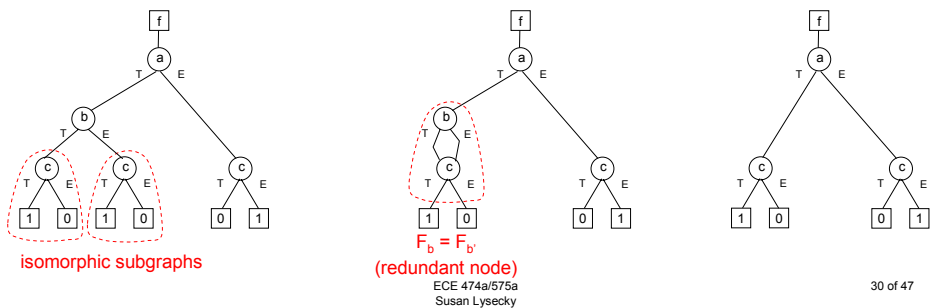
- When building BDDs, result not always reduced (Example 1 - slide 19)
  - We have isomorphic subgraphs, potential for reduction
- Transform a non-reduced BDD into a reduced BDD by iteratively applying
  - Identify isomorphic subgraphs
  - Remove redundant nodes
- A ordered reduced BDD (ROBDD) is unique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

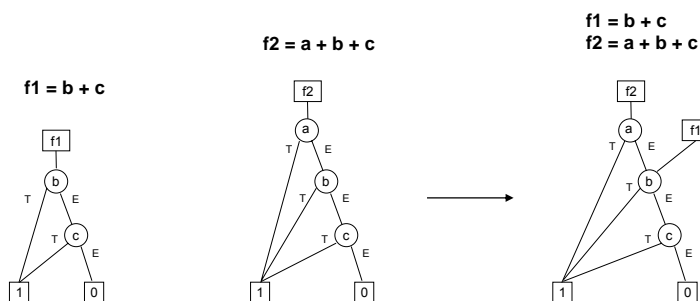


## Representing BDDs

- We said that BDDs can be efficiently implemented
  - Efficient in terms of memory and CPU
- Many BDD Packages Available
  - ABCD, BuDDy, CMU BDD, CrocoPat, CUDD, JavaBDD, Berkeley CAL package, TUD BDD, JDD, JBDD, BDD-PROJECT, BFunc, DDD
- We'll look at high-level design considerations only

## Shared BDDs

- Each node has a function associated with it
- Possible for multiple functions to represent same subexpression
  - Multiple roots sharing same DAG
  - Efficient use of memory



## Unique Table

- Want reduced BDDs
  - Option 1 - Build non-reduced BDD then reduce them in a second pass
  - Option 2 – As we build BDDs guarantee at any time there are no isomorphic subgraphs or redundant nodes in a multi-rooted BDD
- Unique Table
  - Check for existence of node representing desired function before new nodes are added
  - Create dictionary of functions represented in DAC (directed acyclic graph)
  - Implemented as hash table

## What's a dictionary?

- Dictionary
  - Collection of pairs  $(k, v)$  where  $k$  is a key and  $v$  is the value associated with the key
  - No two pairs the same
- Operations performed on dictionaries
  - EMPTY – determine if dictionary empty
  - SIZE – determine number of pairs in dictionary
  - SEARCH – find pair with specified key
  - INSERT – insert pair into dictionary
  - ERASE – delete pair from dictionary
- Duplicate pairs
  - Extend dictionary to have duplicates
    - Permit two or more pairs to have the same key
  - Need some way to eliminate ambiguity in finding/erasing duplicates

Room #	Occupant
ECE 356A	Ten Eyck, B.
ECE 356B	Akoglu, A.
ECE 356C	Lysecky, S.

...

ECE 356J	Sprinkle, J.
----------	--------------

Student ID	Name	Score
12345	Smith, J.	85%
12589	Simpson, L.	92%
20597	Fry, P.	55%

...

89754	Rodriguez, B.	97%
-------	---------------	-----

## What's a dictionary?

- Searching with duplicate pairs
  - Searching is application specific
  - Option 1 – Find any pair with the specified key
    - User just wants a plumber, doesn't care which one
  - Option 2 – Find all pairs with the specified key
    - Instructor wants to know who's in group 7
- In terms of BDDs, we won't have duplicates
  - If the key exists, we simply use the subgraph
  - If the key doesn't exist, we create the subgraph

Business Type	Business Name	Phone Number
Plumber	Mr. Rooter Plumbing	(800) 863-6098
Plumber	Rescue Rooter	(866) 851-1202
Plumber	Senor Rooter	(520) 889-2900
...		
Pizza	Blackjack Pizza	(520) 741-2121

Group ID	Name	Email
1	Smith, J.	smith@aol.com
4	Simpson, L.	siml@gmail.com
7	Fry, P.	pfry@email.arizona.edu
...		
7	Rodriguez, B.	bbr@hotmail.com

## Directed Address Table

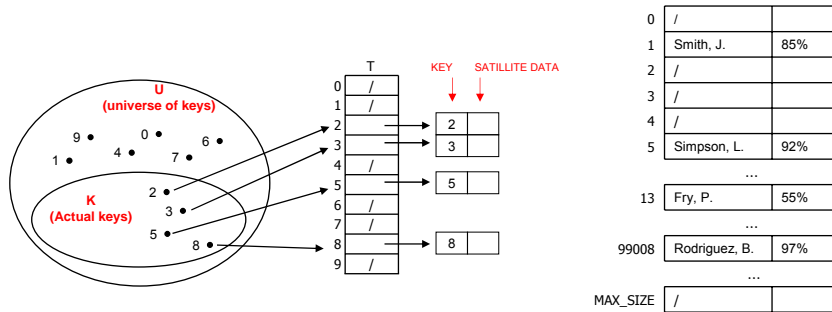
- Direct Address Table (Direct Address Hash Table)
  - Data structure to implement a dictionary
  - Given a universe of keys, each slot in the table (Directed Address Table) corresponds to a key
  - If a (key, value) pair exists, then  $T[k]$  contains the actual data value or a pointer to the (key, value) pair
  - If no element with key  $k$ ,  $T[k] = \text{NULL}$

Student ID	Name	Score
00001	Smith, J.	85%
00005	Simpson, L.	92%
00013	Fry, P.	55%
...		
99008	Rodriguez, B.	97%

0	/	
1	Smith, J.	85%
2	/	
3	/	
4	/	
5	Simpson, L.	92%
...		
13	Fry, P.	55%
...		
99008	Rodriguez, B.	97%
...		
MAX_SIZE	/	

## Directed Address Table

- Works well when universe of keys are relatively small
- Depending on actual keys, we may waste space in memory (lots of empty cells)

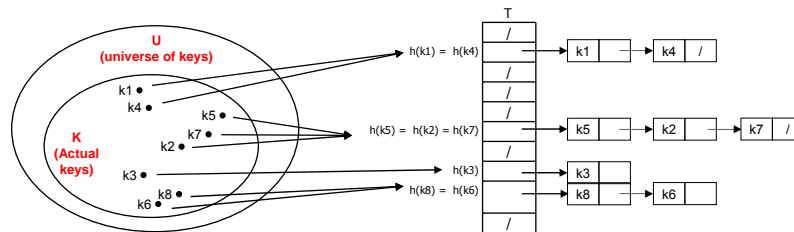


ECE 474a/575a  
Susan Lysecky

37 of 47

## Hash Table

- Hash Table
  - Instead of storing key  $k$  into slot  $k$ , store at  $h(k)$  where  $h$  is the hashing function
  - Design hashing function to reduce number of array indices that must be handled
- What happens when 2 keys map to same slot? Collisions!
  - Avoid collisions by designing hash functions to make sure keys map to different locations, if  $|U| > \text{size of table}$  difficult if not impossible to do this
  - Chaining – okay to map keys to same slot, put all elements into a linked list

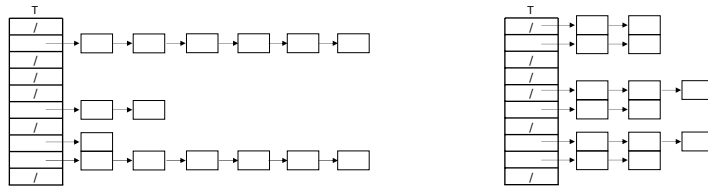


ECE 474a/575a  
Susan Lysecky

38 of 47

## Hashing Function

- A hash function maps a key to an index within in a range
- Desirable properties
  - Simple and quick to calculate
  - Even distribution, avoid collision as much as possible



ECE 474a/575a  
Susan Lysecky

39 of 47

## Hashing Function

- Division Method
  - $h(k) = k \bmod m$  //  $m = \text{size of } T$
  - key mapped by finding remainder
- Multiplication Method
  - $h(k) = \lfloor m(kA \bmod 1) \rfloor$
  - key multiplied by constant value of  $A$  ( $0 < A < 1$ )
  - fractional part of  $kA$  is multiplied by  $m$
  - floor taken to remove fraction part
- Universal Hashing Method
  - Previous methods based on key
  - Given bad set of keys, functions may still not provide uniform hashing
  - Select hashing function from class of hash functions randomly at run time
- Many other hashing methods exist

$$m = 12; k = 100$$

$$h(100) = 100 \bmod 12$$

$$h(100) = 4$$

$$m = 1000; k = 123456; A = 0.61803$$

$$h(123456) = \lfloor 1000(123456(0.61803) \bmod 1) \rfloor$$

$$h(123456) = \lfloor 1000(76299.51168 \bmod 1) \rfloor$$

$$h(123456) = \lfloor 1000(0.51168 \bmod 1) \rfloor$$

$$h(123456) = \lfloor 511.68 \rfloor$$

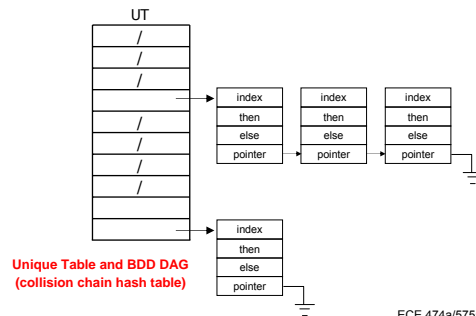
$$h(123456) = 511$$

ECE 474a/575a  
Susan Lysecky

40 of 47

## Unique Table Represented as Hash Table

- BDD Package needed a Unique Table
  - Use the hash table
  - Each node in BDD is the triple (F, G, H)
    - If F then G else H
    - Just like Shannon Expansion,  $F = xF_x + x'F_{x'}$
  - Check for existence of node representing desired function before new nodes are added - look in hash table for matching index



ECE 474a/575a  
Susan Lysecky

41 of 47

## Strong Canonicity

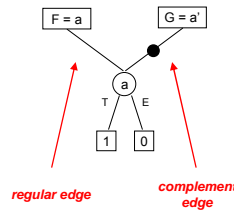
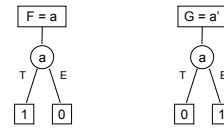
- Using the unique table, two equivalent functions share the same subgraph
- Checking for equivalence is as simple as checking that pointers associated with two functions are identical
- Benefit is constant-time equivalence check

ECE 474a/575a  
Susan Lysecky

42 of 47

## Attributed Edges

- BDDs for  $f$  and  $f'$  are very similar
  - Difference is value of the leaves
- Idea – can we use the same subgraph to represent both functions?
  - Yes! If we can remember the function in the multi-rooted subgraph is the complement
- How?
  - Attach an edge with complement attribute called complement edge
  - Edges without attribute are regular edges
- Benefits
  - Reduced the memory requirement!
  - Complement operation can be performed in constant time

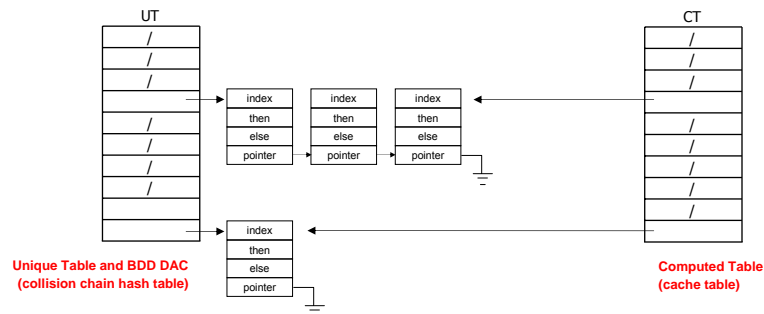


## Computed Table

- Computed Table
  - Speed improvement device
  - Inclusion of CT has resulted in speedups in building BDDs as much as 900%!
- Unique Table vs. Computed Table
  - Unique table verifies if a node exist (to ensure isomorphic subgraphs)
  - Computed table detects if function recently computed (avoid re-computation of same function)

## Computed Table

- Computed Table
  - Contains a triplet (F, G, H) from previous computation
  - Implemented as a "Cache-Based Hash Table" in many BDD packages
  - Does NOT use collision chains
    - If a collision occurs current entry is discarded and new result is inserted



ECE 474a/575a  
Susan Lysecky

45 of 47

## Memory Management

- Garbage Collection
  - Typical application builds and disposes of many BDDs
  - Must efficiently utilize memory
  - Instead of immediately freeing nodes no longer utilized, periodically traverse data structure to recover unused memory
- Dynamic Re-ordering
  - BDDs tend to grow as they are manipulated
    - May run out of memory
  - Variable ordering matters!
    - Don't know best ordering ahead of time so try dynamic re-ordering of BDD variables to see if we can reduce size

ECE 474a/575a  
Susan Lysecky

46 of 47

## 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
  - Hash tables
- What about algorithms to manipulating BDDs?
  - See handwritten notes (Lecture 11B)