EECS 219C: Computer-Aided Verification

Boolean Satisfiability Solving III & Binary Decision Diagrams

Sanjit A. Seshia

EECS, UC Berkeley

Acknowledgments: Lintao Zhang

DLL Algorithm Pseudo-code

Preprocess

Branch

Propagate implications of that branch and deal with conflicts

```
DLL_iterative()
{
    status = preprocess();
    if (status!=UNKNOWN)
        return status;
    while(1) {
        decide_next_branch();
        while (true)
        {
            status = deduce();
            if (status == CONFLICT)
            {
                blevel = analyze_conflict();
                if (blevel < 0)
                    return UNSATISFIABLE;
                else
                    backtrack(blevel);
            }
            else if (status == SATISFIABLE)
                return SATISFIABLE;
            else break;
        }
    }
}
```

DLL Algorithm Pseudo-code

```
DLL_iterative()
{
    status = preprocess();
    if (status!=UNKNOWN)
        return status;
    while(1) {
        decide_next_branch();
        while (true)
        {
            status = deduce();
            if (status == CONFLICT)
            {
                blevel = analyze_conflict();
                if (blevel < 0)
                    return UNSATISFIABLE;
                else
                    backtrack(blevel);
            }
            else if (status == SATISFIABLE)
                return SATISFIABLE;
            else break;
        }
    }
}
```

Main Steps:

Pre-processing

Branching

Unit propagation (apply unit rule)

Conflict Analysis & Backtracking

S. A. Sesh

3

A Classification of SAT Algorithms

- Davis-Putnam (DP)
  - Based on **resolution**
- Davis-Logemann-Loveland (DLL/DPLL)
  - Search-based
  - Basis for current most successful solvers
- Stalmarck's algorithm [optional reading]
  - More of a "breadth first" search, proprietary algorithm
- Stochastic search
  - Local search, hill climbing, etc.
  - Unable to prove unsatisfiability (incomplete)

S. A. Seshia

4

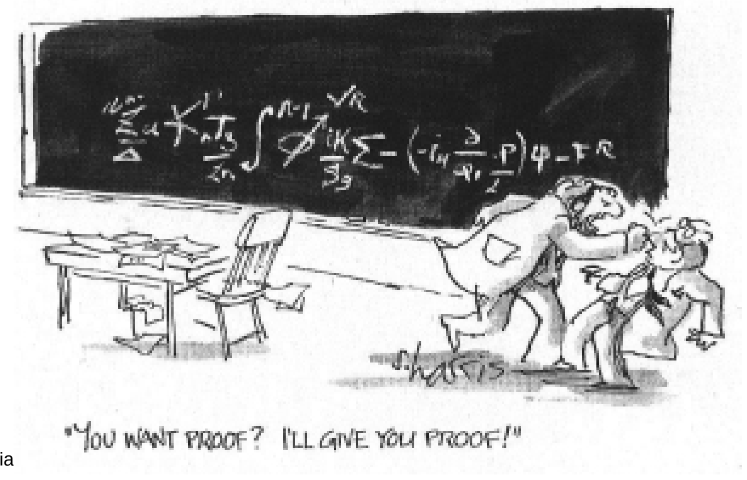
Proof Generation

- SAT Solvers are complicated pieces of software
- What if there is a bug in the solver? How can we check its output?

S. A. Seshia 5

Proof Generation

- If the SAT solver returns "satisfiable", we can check that solution by evaluating the circuit
- If it returns "unsatisfiable", what then?



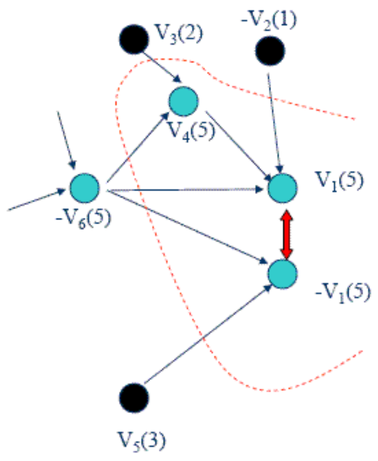
S. A. Seshia 6

Proof

- Starting from facts (clauses), the SAT solver has presumably derived "unsatisfiable" (the empty clause)
- So there must be a way of going step-by-step from input clauses to the empty clause using rules
  - In fact, there's only one rule: **resolution**

S. A. Seshia 7

Resolution as a Cut in Implication Graph



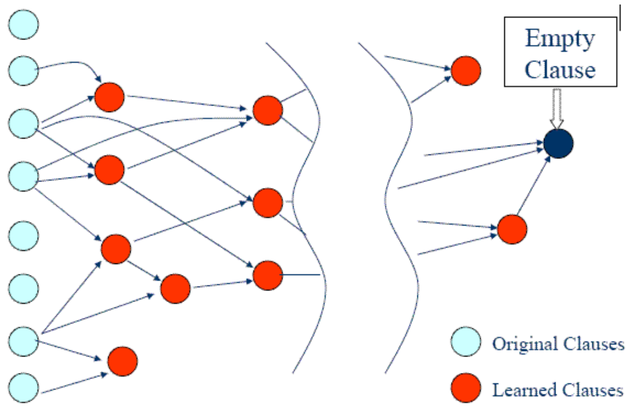
$(V_2 + V_3' + V_5' + V_6)$

$(V_3' + V_6 + V_4)$
$(V_6 + V_5' + V_1')$
$(V_2 + V_4' + V_6 + V_1)$
$(V_2 + V_4' + V_6 + V_5')$
$(V_2 + V_3' + V_5' + V_6)$

S. A. Seshia 8

Resolution Graph

- Nodes are clauses
- Edges are applications of resolution



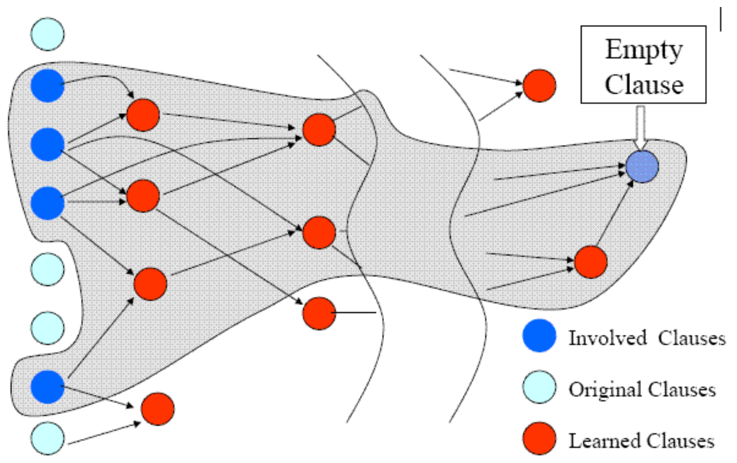
S. A. Seshia 9

Proof Checker

- Given resolution graph, how to check it?
- Traverse it, checking that each node is correctly obtained from its predecessor nodes using resolution

S. A. Seshia 10

Unsatisfiable Core



S. A. Seshia 11

Incremental SAT Solving

- Suppose you have not just one SAT problem to solver, but many "slightly differing" problems over the same variables
- Can we re-use the search over many problems?
  - i.e. perform only "incremental" work

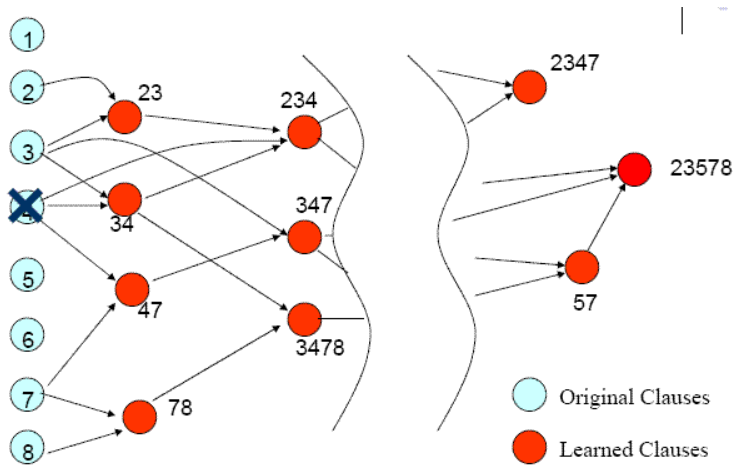
S. A. Seshia 12

Operations Needed

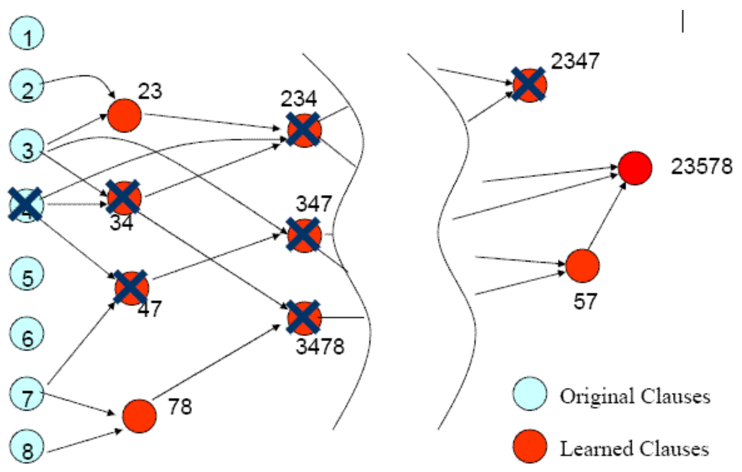
1. Adding clauses
2. Deleting clauses

- Which is easy and which is hard?
  - If previous problem is unsat, how does an operation change it?
  - If previous is sat?

Deleting Clauses



Deleting Clauses



S. A. Seshia 15

Engineering Issues

- Too expensive to traverse graph
- Instead, group original clauses into groups
- Each derived clause belongs to all groups that it is resolved from
  - Implement with bit-vector

S. A. Seshia 16

Binary Decision Diagrams

S. A. Seshia 17

Boolean Function Representations

- Syntactic: e.g.: CNF, DNF, Circuit
- Semantic: e.g.: Truth table, Binary Decision Tree, BDD

S. A. Seshia 18

Reduced Ordered BDDs

- Invented by Randal E. Bryant in mid-80s
  - IEEE Transactions on Computers 1986 paper is one of the most highly cited papers in EECS
- Useful data structure to represent Boolean functions
  - Applications in synthesis, verification, program analysis, planning, …
- Commonly known simply as BDDs
- Many variants of BDDs have proved useful in other tasks
- Links to coding theory (trellises), etc.

S. A. Seshia 19

Cofactors

- A Boolean function F of n variables $x_1, x_2, \dots, x_n$

$$F : \{0,1\}^n \rightarrow \{0,1\}$$

- Suppose we define new Boolean functions of n-1 variables as follows:

$$F_{x_1}(x_2, \dots, x_n) = F(1, x_2, x_3, \dots, x_n)$$
$$F_{x_1'}(x_2, \dots, x_n) = F(0, x_2, x_3, \dots, x_n)$$

- F_{x_1} and $F_{x_1'}$ are called cofactors of F. F_{x_1} is the positive cofactor, and $F_{x_1'}$ is the negative cofactor

S. A. Seshia 20

Shannon Expansion

- $F(x_1, \ldots, x_n) = x_i \,.\, F_{x_i} + x_i' \,.\, F_{x_i'}$

- Proof?

S. A. Seshia 21

Shannon expansion with many variables

- $F(x, y, z, w) =$
  $xy\ F_{xy} + x'y\ F_{x'y} + xy'\ F_{xy'} + x'y'\ F_{x'y'}$

S. A. Seshia 22

Properties of Cofactors

- Suppose you construct a new function H from two existing functions F and G: e.g.,
  - H = F'
  - H = F.G
  - H = F + G
  - Etc.
- What is the relation between cofactors of H and those of F and G?

S. A. Seshia 23

Very Useful Property

- Cofactor of NOT is NOT of cofactors
- Cofactor of AND is AND of cofactors
- …
- Works for any binary operator

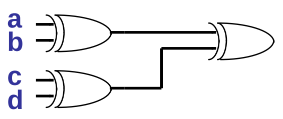
S. A. Seshia 24

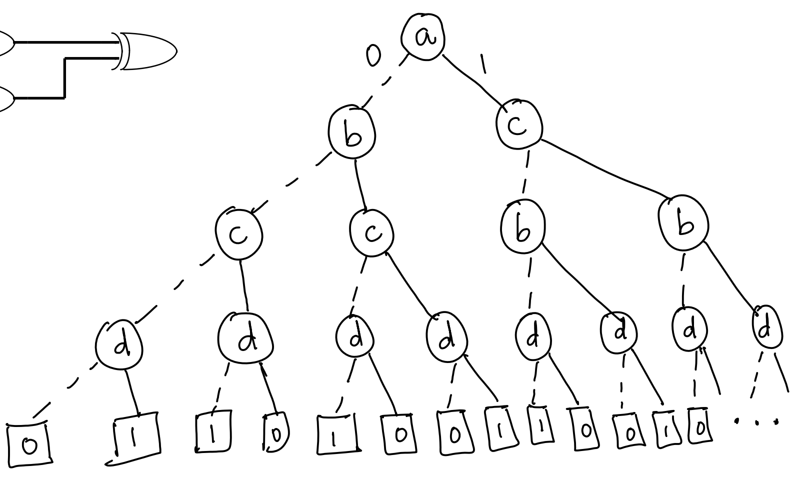
BDDs from Truth Tables

$$\begin{tikzcd}
\text{Truth Table} \arrow[d] \\
\text{Binary Decision Tree} \arrow[d] \\
\text{Binary Decision Diagram (BDD)} \arrow[d] \\
\text{Ordered Binary Decision Diagram (OBDD)} \arrow[d] \\
\text{Reduced Ordered Binary Decision Diagram (ROBDD, simply called BDD)}
\end{tikzcd}$$

S. A. Seshia 25

Example: Odd Parity Function

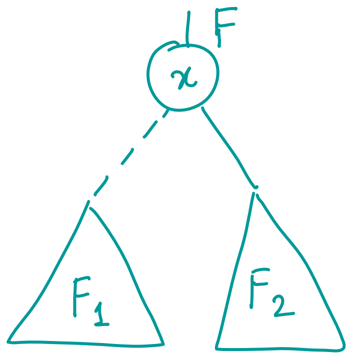




Binary Decision Tree

S. A. Seshia 26

Nodes & Edges



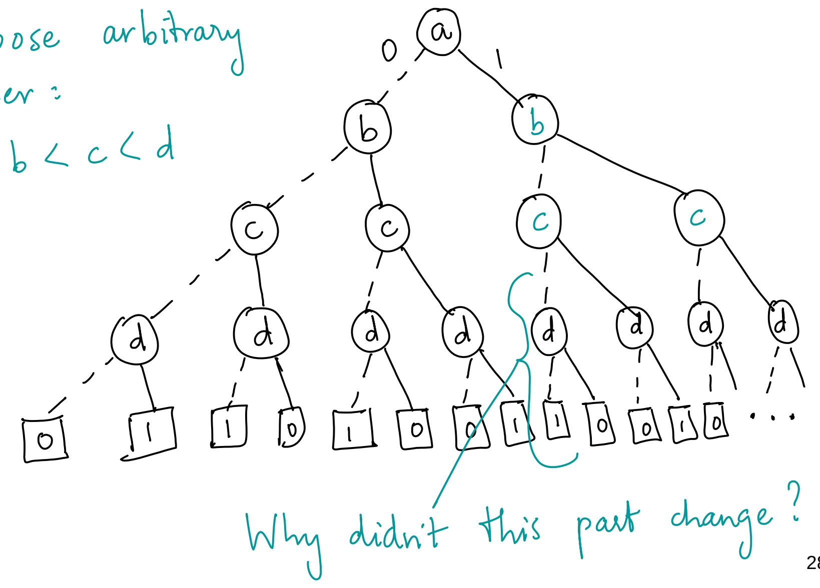
- How is F related to x, F_1, F_2 ?

S. A. Seshia 27

Ordering

Impose arbitrary order :

$a < b < c < d$



S. A. Seshia 28

Reduction

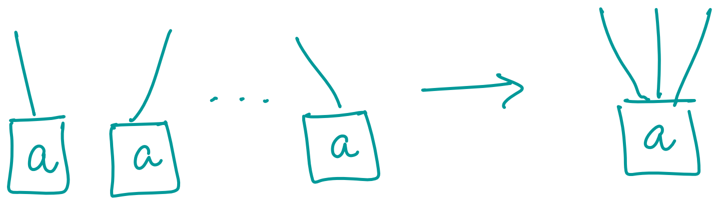
- Identify Redundancies

- 3 Rules:

1. Merge equivalent leaves
2. Merge isomorphic nodes
3. Eliminate redundant tests

S. A. Seshia 29

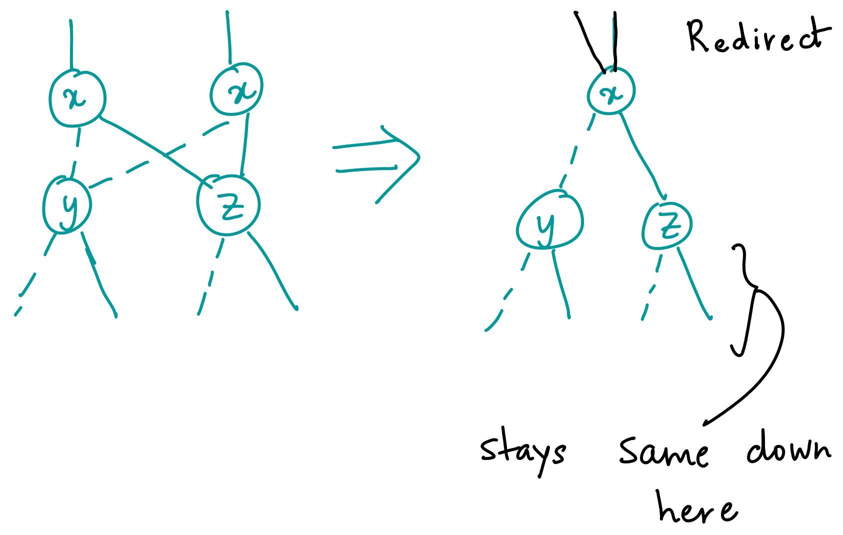
Merge Equivalent Leaves



"a" is either 0 or 1

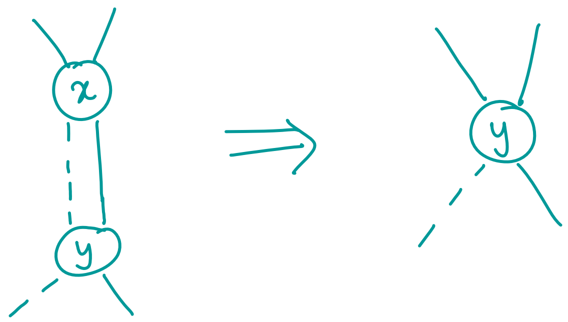
S. A. Seshia 30

Merge Isomorphic Nodes



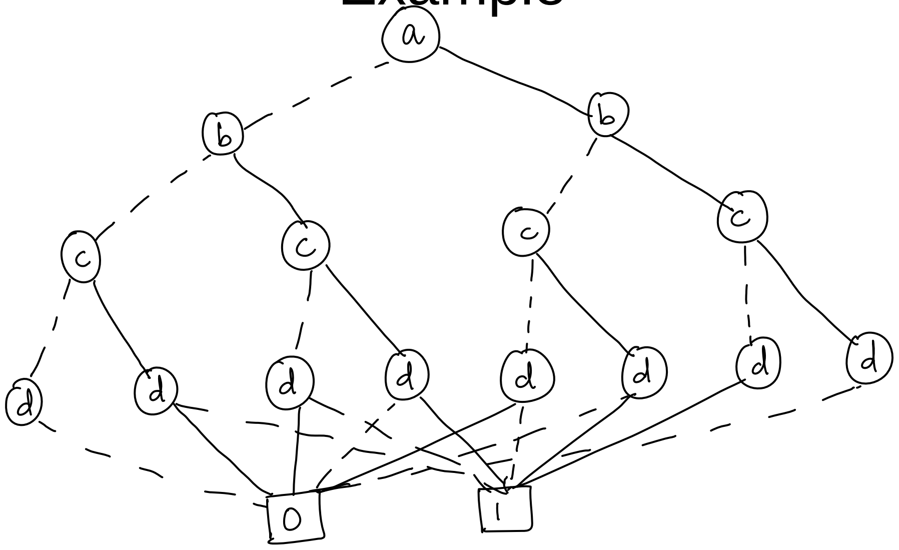
S. A. Seshia 31

Eliminate Redundant Tests

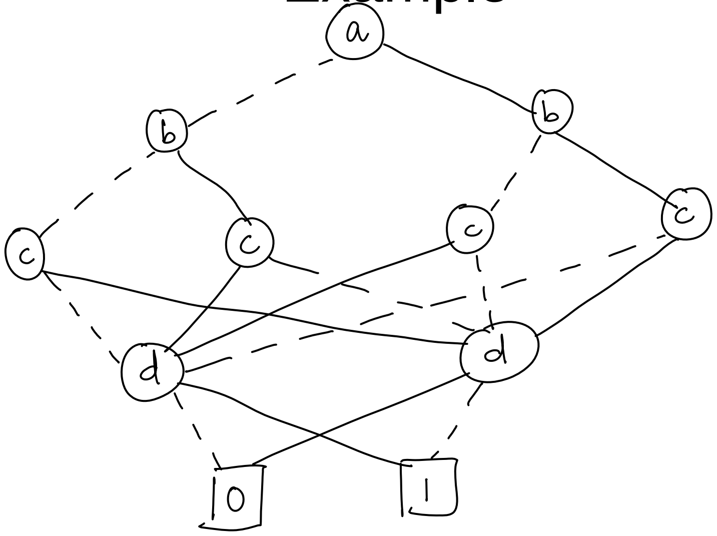


S. A. Seshia 32

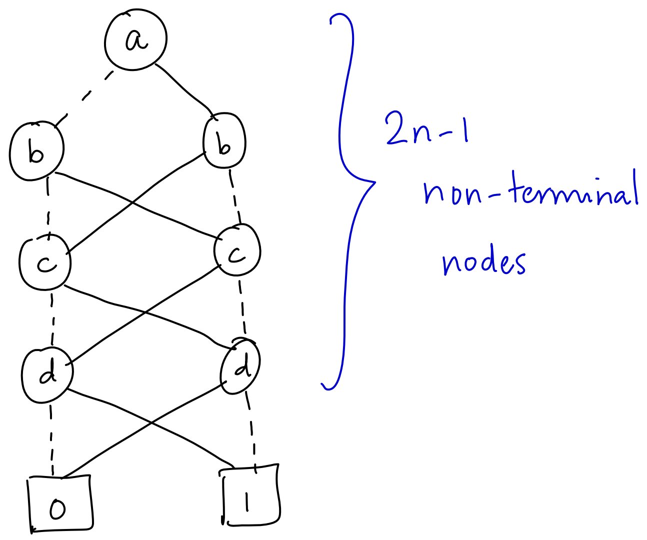
Example



Example

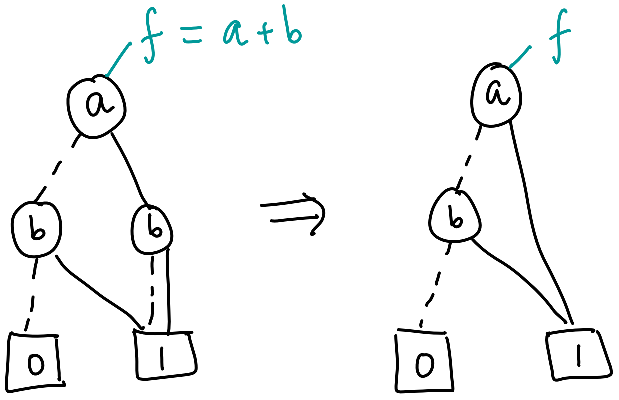


Final ROBDD for Odd Parity Function



S. A. Seshia 35

Example of Rule 3



S. A. Seshia 36

What can BDDs be used for?

- Uniquely representing a Boolean function
  - And a Boolean function can represent sets
- Satisfiability solving!

S. A. Seshia 37

(RO)BDDs are canonical

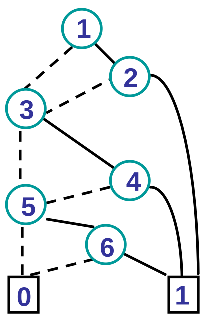
- Theorem (R. Bryant): If G, G' are ROBDD's of a Boolean function f with k inputs, using same variable ordering, then G and G' are identical.

S. A. Seshia 38

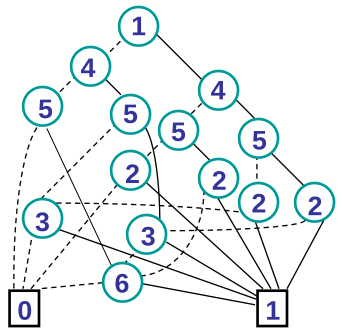
Sensitivity to Ordering

- Given a function with n inputs, one input ordering may require exponential # vertices in ROBDD, while other may be linear in size.
- Example: $f = x_1 x_2 + x_3 x_4 + x_5 x_6$

$x_1 < x_2 < x_3 < x_4 < x_5 < x_6$



$x_1 < x_4 < x_5 < x_2 < x_3 < x_6$



S. A. Seshia 39

Creating BDDs

- Idea: Build a few core operators and define everything else in terms of those

Advantage:
• Less programming work
• Easier to add new operators later by writing “wrappers”

S. A. Seshia 40

Core Operators

- Just two of them!

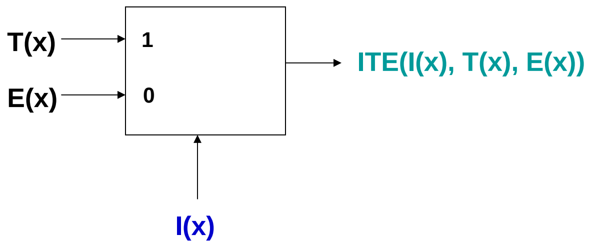
1. Restrict(Function F, variable v, constant k)
   - Shannon cofactor of F w.r.t. v=k

2. ITE(Function I, Function T, Function E)
   - "if-then-else" operator

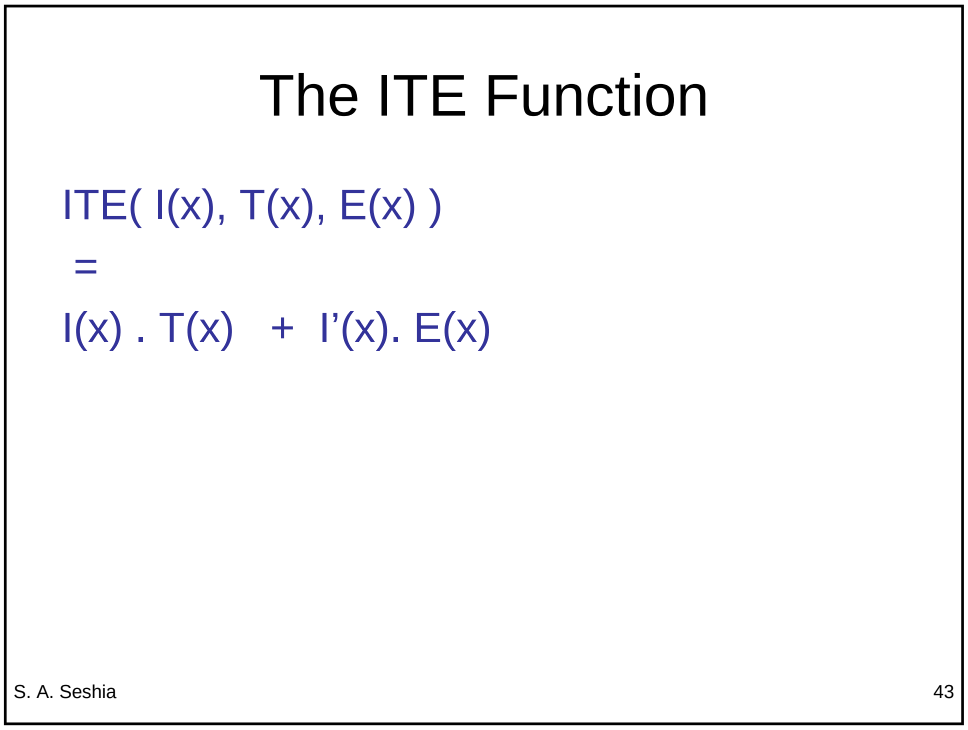
S. A. Seshia 41

ITE

- Just like:
  – "if then else" in a programming language
  – A mux in hardware
- ITE(I(x), T(x), E(x))
  – If I(x) then T(x) else E(x)



S. A. Seshia 42



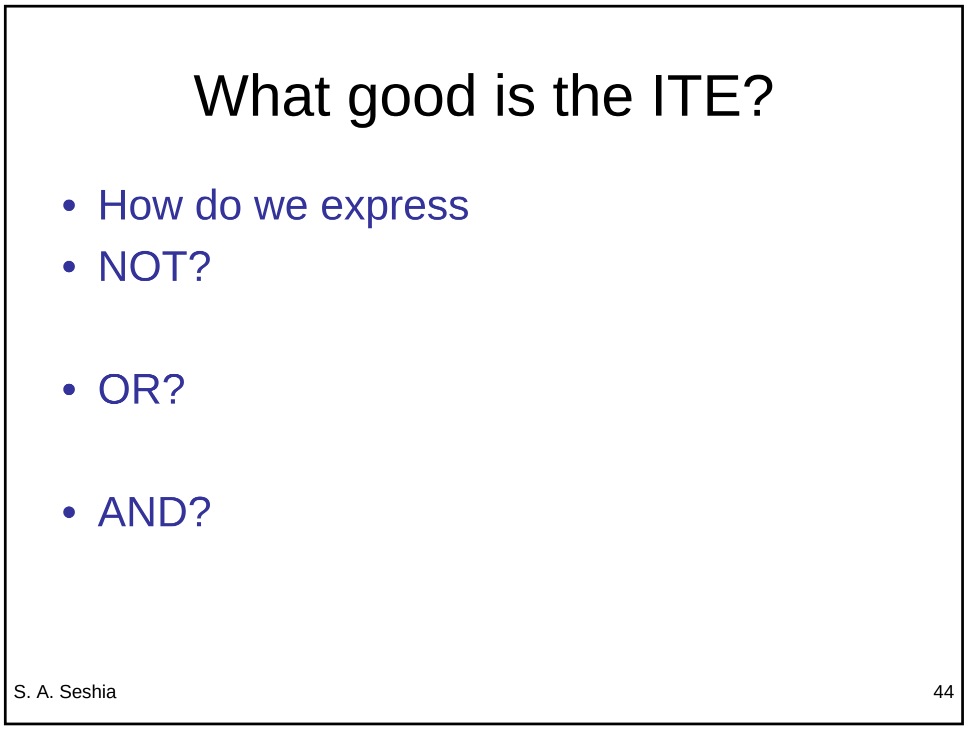
The ITE Function

ITE(I(x), T(x), E(x))

=

I(x) . T(x) + I'(x). E(x)

S. A. Seshia 43



What good is the ITE?

- How do we express
- NOT?
- OR?
- AND?

S. A. Seshia 44

How do we implement ITE?

- Divide and conquer!
- Use Shannon cofactoring…
- Recall: Operator of cofactors is Cofactor of operators…

ITE Algorithm

```
ITE (bdd I, bdd T, bdd E) {
   if (terminal case) { return computed result;
   }
   else { // general case
      Let x be the topmost variable of I, T, E;
      PosFactor = ITE($I_x$ , $T_x$ , $E_x$) ;
      NegFactor = ITE($I_{x'}$ , $T_{x'}$ , $E_{x'}$);
      R = new node labeled by x;
      R.low = NegFactor; // set 0-child
      R.high = PosFactor; // set 1-child
      Reduce(R); // apply reduction rules
      return R;
}
```

Terminal Cases

- ITE(1, T, E) =
- ITE(0, T, E) =
- ITE(I, T, T) =
- ITE(I, 1, 0) =
- …

S. A. Seshia 47

General Case

- Still need to do cofactor (Restrict)
- Easy operation in the case of ITE
  - We are cofactoring w.r.t. the top variable only
  - Cofactoring w.r.t. an internal variable is trickier

S. A. Seshia 48

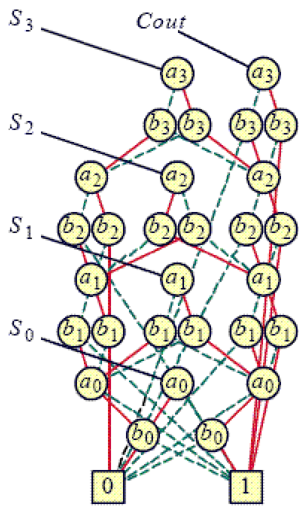
Practical Issues

- Previous calls to ITE are cached
  - "memoization"
- Every BDD node created goes into a "unique table"
  - Before creating a new node R, look up this table
  - Avoids need for reduction

S. A. Seshia 49

Sharing: Multi-Rooted DAG

- BDD for 4-bit adder
- Each output bit (of the sum & carry) is a distinct rooted BDD
- But they share sub-DAGs



S. A. Seshia 50

More on BDDs

- Circuit width and bounds on BDD size
- Dynamically changing variable ordering
- Some BDD variants

Bounds on BDD Size: Warm-up

- Suppose the number of nodes at any level in a BDD is bounded above by B
- Then, what is an upper bound on the total number of nodes in the BDD?

Cross-section of a BDD at level i

- Suppose a BDD represents Boolean function $F(x_1, x_2, \ldots, x_n)$ with variable order $x_1 < x_2 < \ldots < x_n$
- Size of cross section of the BDD at level i is the number of distinct Boolean functions F' that *depend on* x_i given by

  $F'(x_i, x_{i+1}, \ldots, x_n) = F(v_1, v_2, \ldots, v_{i-1}, x_i, \ldots, x_n)$

  for some Boolean constants v_i's (in {0,1})

Circuit Width

- Consider a circuit representation of a Boolean function F
- Impose a linear order on the gates of the circuit
  - Primary inputs and outputs are also considered as "gates" and primary output is at the end of the ordering
  - Forward cross section at a gate g: set of wires going from output of g_1 to input of g_2 where $g_1 \leq g < g_2$
  - Similarly define reverse cross section: set of wires going from output of g_1 to input of g_2 where $g_2 \leq g < g_1$
  - Forward width (w_f): maximum forward cross section size
  - Similarly, reverse width w_r

BDD Upper Bounds from Circuit Widths

- Theorem: Let a circuit representing F with ***n*** variables have forward width w_f and reverse width w_r for some linear order L on its gates. Then, there is a BDD representing F of size bounded above by

$$n.2^{w_f \cdot 2^{w_r}}$$

S. A. Seshia 55

BDD Ordering in Practice

- If we can derive a small upper bound using circuit width, then that's fine
  - Use the corresponding linear order on the variables
- What if we can't?

- There are many BDD variable ordering heuristics around, but the most common way to deal with variable ordering is to start with something "reasonable" and then swap variables around to improve BDD size
  - DYNAMIC VARIABLE REORDERING → SIFTING

S. A. Seshia 56

Sifting

- Dynamic variable re-ordering, proposed by R. Rudell
- Based on a primitive "swap" operation that interchanges x_i and x_{i+1} in the variable order
  - Key point: the swap is a local operation involving only levels i and i+1
- Overall idea: pick a variable x_i and move it up and down the order using swaps until the process no longer improves the size
  - A "hill climbing" strategy

Some BDD Variants

- Free BDDs (FBDDs)
  - Relax the restriction that variables have to appear in the same order along all paths
  - How can this help?
  - Is it canonical?

Some BDD Variants

- MTBDD (Multi-Terminal BDD)
  - Represents function of Boolean variables with non-Boolean value (integer, rational)
    - E.g., input-dependent delay in a circuit, transition probabilities in a Markov chain
  - Similar reduction / construction rules to BDDs

Some BDD packages

- CUDD – from Colorado University, Fabio Somenzi's group
- BuDDy – from IT Univ. of Copenhagen