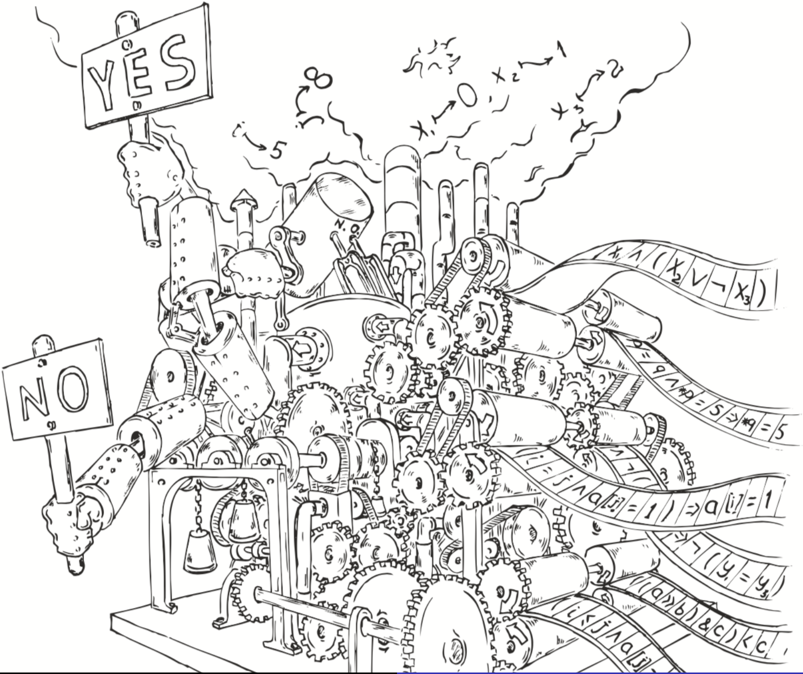

YES
NO

SAT/SMT solvers
11. Binary Decision Diagrams

Roman Kholin

Lomonosov Moscow State University

Moscow, 2023

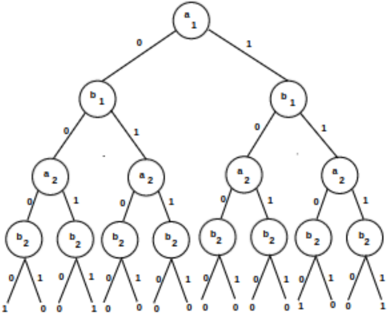
Binary Decision Trees

A binary decision tree is a rooted, directed tree with two types of vertices, terminal vertices and nonterminal vertices.
Each nonterminal vertex v is labeled by a variable $var(v)$ and has two successors:

- $low(v)$ corresponding to the case where the variable v is assigned 0
- $high(v)$ corresponding to the case where the variable v is assigned 1

Each terminal vertex v is labeled by $value(v)$ which is either 0 or 1

$f(a_1, a_2, b_1, b_2) = (a_1 \iff b_1) \wedge (a_2 \iff b_2)$



There is usually a lot of redundancy in such trees

There are eight subtrees with roots labeled by b_2, but only three are distinct

Binary Decision Diagrams

- Binary decision diagram is a rooted, directed acyclic graph with two types of vertices, terminal vertices and nonterminal vertices
- Each nonterminal vertex v is labeled by a variable $var(v)$ and has two successors, $low(v)$ and $high(v)$
- Each terminal vertex is labeled by either 0 or 1

Example

$f = x_1 x_2 + x_4$
$(x_1 \wedge x_2 \vee x_4)$

Odd parity function

$x_1 x_2 x_3 x_4$

$\neg, \vee, \wedge$

Canonical Form Property

- Such a representation must guarantee that two boolean functions are logically equivalent if and only if they have isomorphic representations
- This simplifies tasks like checking equivalence of two formulas and deciding if a given formula is satisfiable or not

Two binary decision diagrams are isomorphic if there exists a bijection h between the graphs such that:

- terminals are mapped to terminals and nonterminals are mapped to nonterminals
- for every terminal vertex v, $value(v) = value(h(v))$
- for every nonterminal vertex v:
  - $var(v) = var(h(v))$
  - $h(low(v)) = low(h(v))$
  - $h(high(v)) = high(h(v))$

Canonical Form Property

To obtain a canonical representation for boolean functions by placing two restrictions on binary decision diagrams:

- The variables should appear in the same order along each path from the root to a terminal
- There should be no isomorphic subtrees or redundant vertices in the diagram

The first requirement is easy to achieve:

- Impose total ordering $<$ on the variables in the formula
- Require that if vertex u has a nonterminal successor v, then $var(u) < var(v)$

Canonical Form Property

The second requirement is achieved by repeatedly applying three transformation rules that do not alter the function represented by the diagram:

- Remove duplicate terminals: Eliminate all but one terminal vertex with a given label and redirect all arcs to the eliminated vertices to the remaining one
- Remove duplicate nonterminals: If nonterminals u and v have $var(u) = var(v)$, $low(u) = low(v)$ and $high(u) = high(v)$, then eliminate one of the two vertices and redirect all incoming arcs to the other vertex
- Remove redundant tests: If nonterminal vertex v has $low(v) = high(v)$, then eliminate v and redirect all incoming arcs to $low(v)$

The term Ordered Binary Decision Diagram (OBDD) will be used to refer to the graph obtained in this manner

Example

Ordered Binary Decision Diagrams

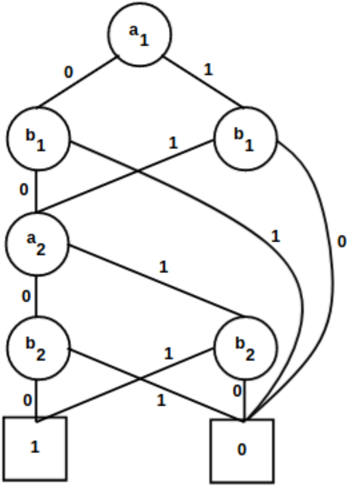
- The canonical form may be obtained by applying the transformation rules until the size of the diagram can no longer be reduced
- It was shown how this can be done by a procedure called Reduce in linear time

If OBDDs are used as a canonical form for boolean functions, then

- checking equivalence is reduced to checking isomorphism between OBDDs
- satisfiability can be determined by checking equivalence with the trivial OBDD that consists of only one terminal labeled by 0

OBDD for Comparator Example

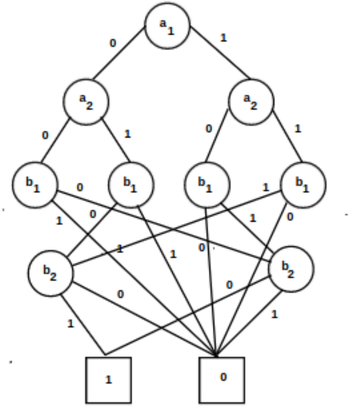
$a_1 < b_1 < a_2 < b_2$



The number of vertices will be $3n + 2$

OBDD for Comparator Example

$a_1 < a_2 < b_1 < b_2$



The number of vertices will be $3 * 2^n - 1$

Logical Operations

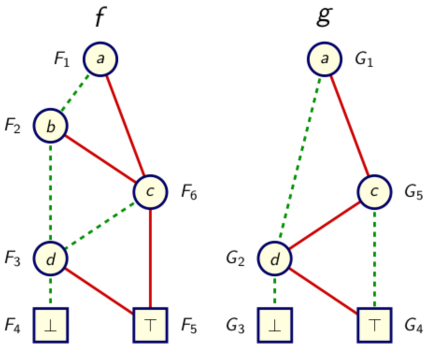
How to get OBDD for $f * g$ (if we know OBDDs for f and g)?
The key idea for efficient implementation of these operations is the Shannon expansion
$f = \neg x \wedge f|_{x=0} \vee x \wedge f|_{x=1}$
Let $*$ be an arbitrary two argument logical operation, and let f and f' be two boolean functions
To simplify the explanation of the algorithm we introduce the following notation:

- v and v' are the roots of the OBDDs for f and f'
- $x = var(v)$ and $x' = var(v')$

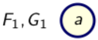
Logical Operations

- If v and v' are both terminal vertices, then $f * f' = value(v) * value(v')$
- If $x = x'$, then we use the Shannon expansion $f * f' = \neg x \wedge (f|_{x=0} * f'|_{x=0}) \vee x \wedge (f|_{x=1} * f'|_{x=1})$
- If $x < x'$, then $f'|_{x=0} = f'|_{x=1} = f'$ since f' does not depend on x. In this case the Shannon Expansion simplifies to $f * f' = \neg x \wedge (f|_{x=0}) \vee x \wedge (f|_{x=1})$
- If $x > x'$ - in the previous manner

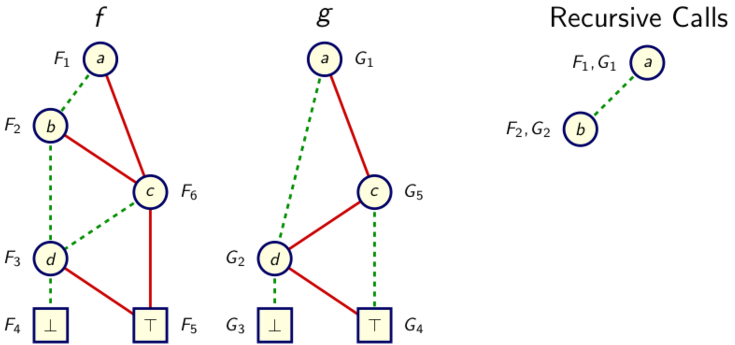
Logical Operations



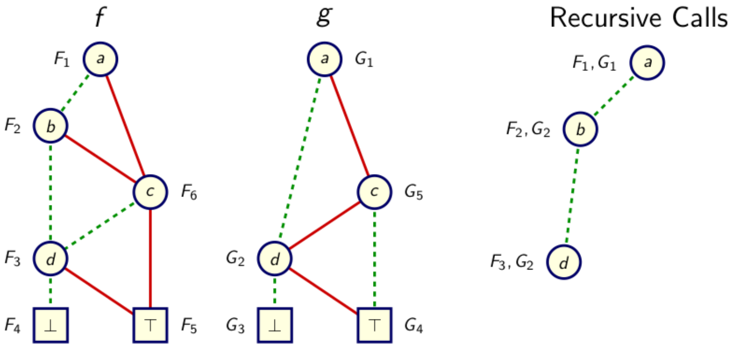
Recursive Calls



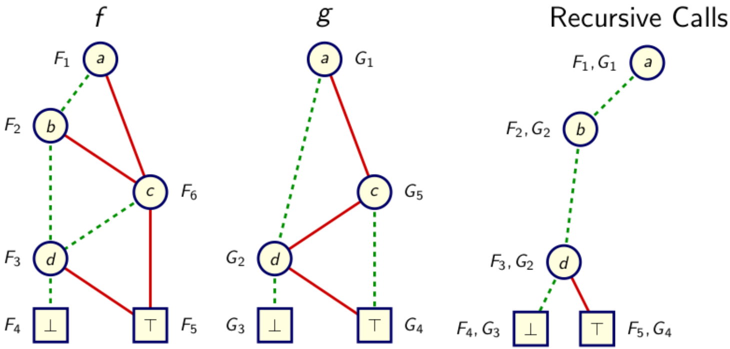
Logical Operations



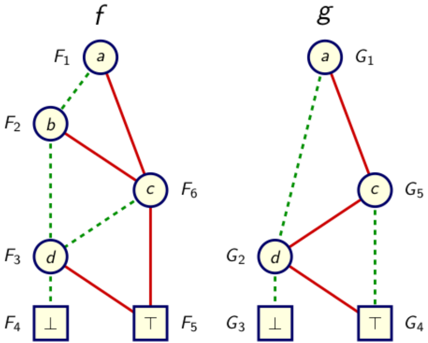
Logical Operations



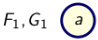
Logical Operations



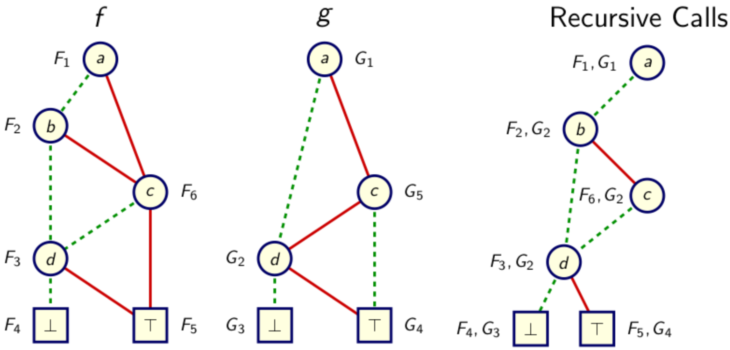
Logical Operations



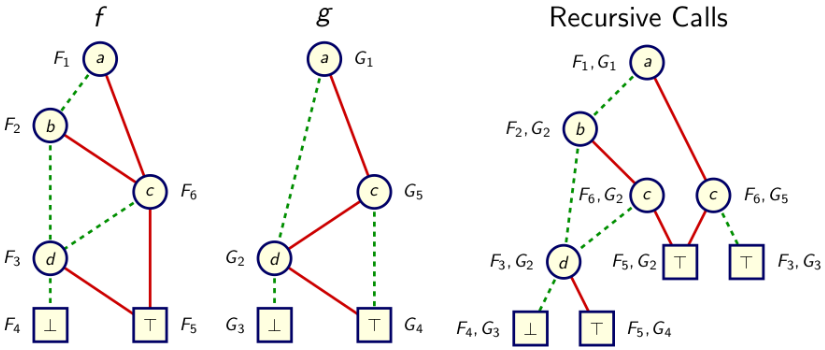
Recursive Calls



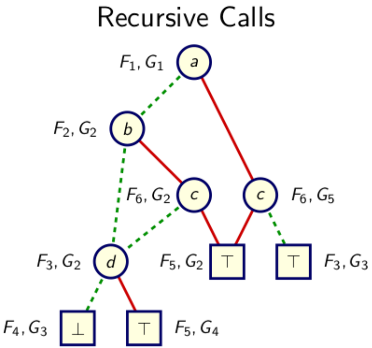
Logical Operations

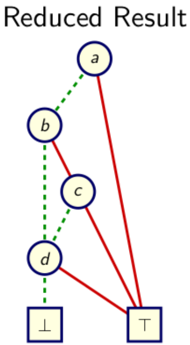


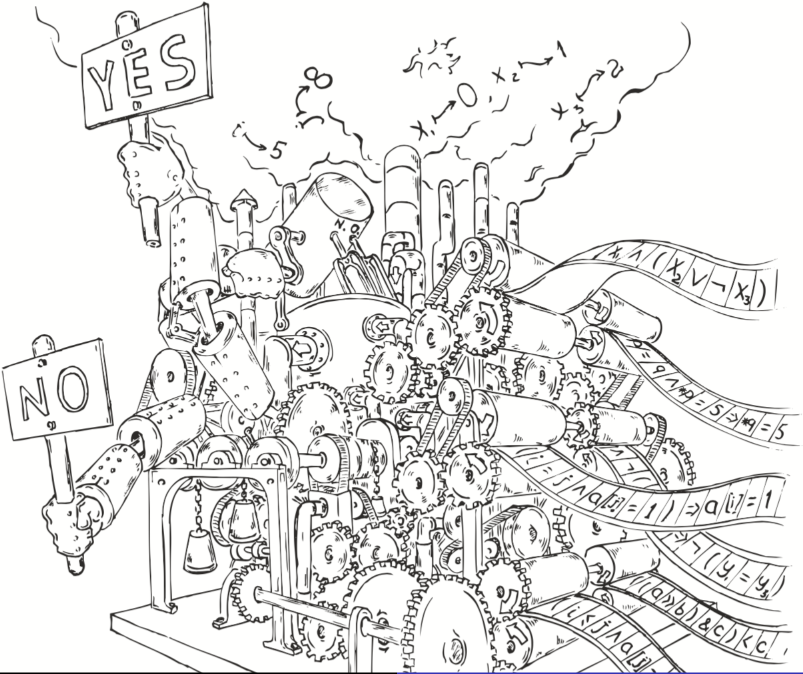
Logical Operations



Logical Operations






YES
NO
N.O.
5
8
0
1
2
(x_2 ∨ ¬x_3)
*a = 5 ⇒ *a = 5
a[i] = 1) ⇒ a[j] = 1
¬(y_1 = y_3)
((a>>b)&c)<c