Computer-Aided Formal Verification (MT 2009)

Binary Decision Diagrams (BDDs)

Daniel Kroening

Oxford University, Computing Laboratory

Version 1.0, 2009

Outline

Motivation

- ✗ The state space of interesting systems is too big, explicit enumeration will fail inevitably.
- ✓ However, the reachable state space isn't random, but follows specific rules
- ✓ We hope to exploit that with data structures that are concise in relevant cases

Binary Decision Diagrams (BDDs)

Randal E. Bryant

- *Binary Decision Diagrams* (BDDs) are a symbolic representation of Boolean functions
- Key idea: specific forms of BDDs are canonical
- [Bryant86] is one of the most-cited papers in computer science

canonical, *a.* (and *n.*)

1. Prescribed by, in conformity with, or having reference to ecclesiastical edict or canon law.
b. **canonical hours**: (a) stated times of the day appointed by the canons for prayer and devotion; (b) the hours (now from 8 a.m. to 3 p.m.) within which marriage can be legally performed in a parish church in England; (c) *transf.*
c. **canonical dress**, etc.; the articles of dress worn by clergy according to canon.
d. canonical obedience: the obedience to be rendered by inferior clergy to the bishop or other ecclesiastical superior, according to the canons.
2. Of or belonging to the canon of Scripture. (Also used of other sacred books.)
3. **canonical epistles**, more particularly, the seven catholic epistles of James, Peter, John, and Jude; also applied to certain epistles of St. Basil, etc. Also quasi-n., ***a canonical*** (obs.) = CANON $n.^2$ 5.
4. *gen.* Of the nature of a canon or rule; of admitted authority, excellence, or supremacy; authoritative; orthodox, accepted; standard.
5. *Math.* Furnishing, or according to, a general rule or formula (see CANON $n.^2$ 3).
6. *Mus.* According to the rules of canon, in canon form.
7. Of or belonging to an ecclesiastical chapter, or to one of its members (see CANON $n.^2$).

Oxford English Dictionary

canonical, *a.* (and *n.*)

1. Prescribed by, in conformity with, or having reference to ecclesiastical edict or canon law.

b. **canonical hours**: (a) stated times of the day appointed by the canons for prayer and devotion; (b) the hours (now from 8 a.m. to 3 p.m.) within which marriage can be legally performed in a parish church in England; (c) *transf*.

c. **canonical dress**, etc.; the articles of dress worn by clergy according to canon.

d. canonical obedience: the obedience to be rendered by inferior clergy to the bishop or other ecclesiastical superior, according to the canons.

2. Of or belonging to the canon of Scripture. (Also used of other sacred books.)

3. **canonical epistles**, more particularly, the seven catholic epistles of James, Peter, John, and Jude; also applied to certain epistles of St. Basil, etc. Also quasi-n., ***a canonical*** (obs.) = CANON *n.*^{2} 5.

4. *gen.* Of the nature of a canon or rule; of admitted authority, excellence, or supremacy; authoritative; orthodox, accepted; standard.

5. *Math.* Furnishing, or according to, a general rule or formula (see CANON *n.*^{2} 3).

6. *Mus.* According to the rules of canon, in canon form.

7. Of or belonging to an ecclesiastical chapter, or to one of its members (see CANON *n.*^{2}).

Oxford English Dictionary

Explicit vs. Symbolic Representations of Functions

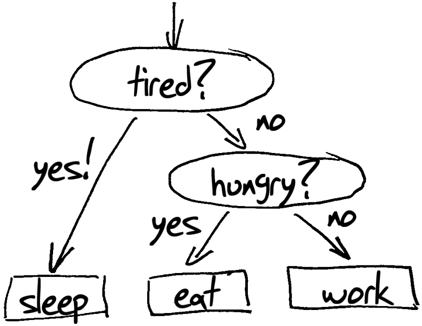
x	y	z	$f(x,y,z)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$$f(x,y,z) = x \leftrightarrow (y \wedge z)$$

Explicit
Obviously gets large

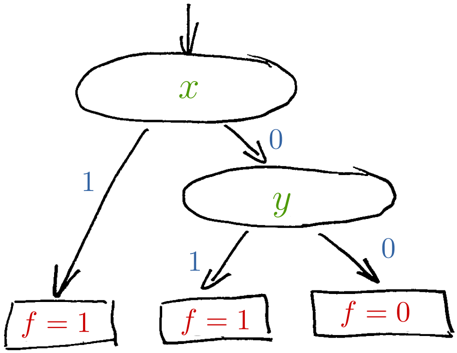
Symbolic
Potentially smaller

Decision Diagrams



- Distinguishes terminal from non-terminal nodes
- The edges are labeled with decisions
- The sink nodes (squares) are labeled with the outcome

Decision Diagrams for Functions

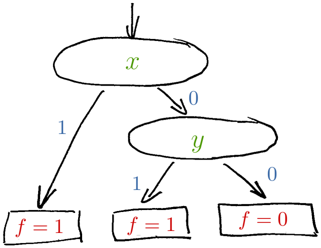


- This encodes $f(x, y) = x \vee y$
- Inner nodes: a variable v
- Out-edges: value for v
- Terminal nodes: function value

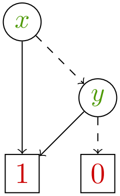
BDD

- represents Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ (n = number of variables)
- as directed acyclic graph (DAG)
  - one root, two leaves (0 and 1), two (colored) edges at inner nodes
- as finite automaton (but no cycles)
  - accepts subset of $\{0,1\}^n$, 1 as accepting state, 0 as dead-end state
- as branching program (but loop-free)
  - `<label>:  if <var> goto <label> else goto <label>`
  - `<label>:  return [ 0 | 1 ]`

Drawing BDDs

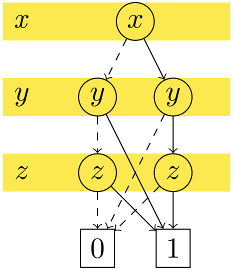


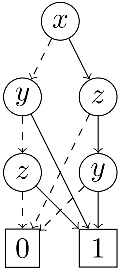
$\longrightarrow$



- A solid edge (“high edge”) means the variable is 1
- A dashed edge (“low edge”) means the variable is 0

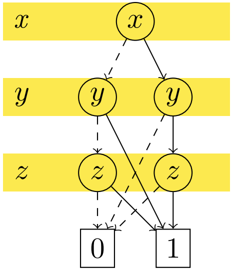
Ordered BDDs

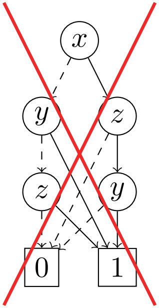




- Assign arbitrary total ordering to variables, e.g., $x < y < z$.
- Variables must appear in that order along all paths.

Ordered BDDs





- Assign arbitrary total ordering to variables, e.g., $x < y < z$.
- Variables must appear in that order along all paths.

Reduced Ordered BDDs (RO-BDDs)

A reduced ordered BDD (RO-BDD) is obtained from an ordered BDD by applying three rules:

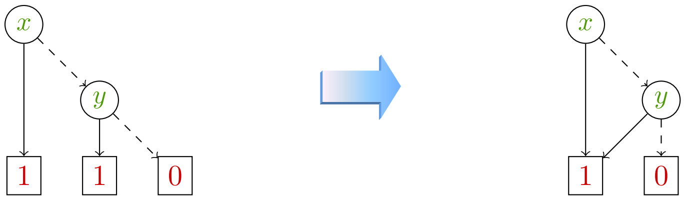
$\#1$: Merge equivalent leaf nodes

$\#2$: Merge isomorphic nodes

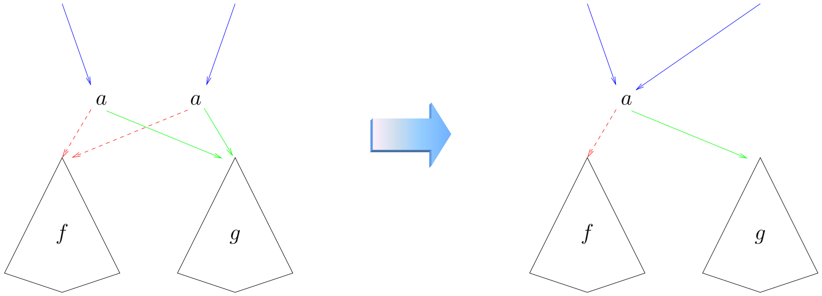
$\#3$: Eliminate redundant tests

Reduction #1: Equivalent Leaf Nodes

This one is trivial:



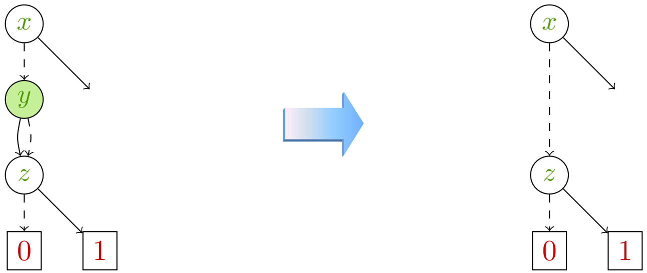
Reduction #2: Isomorphic Nodes



Merge nodes with the same variable and the same children

Reduction #3: Redundant Tests

Eliminate nodes where both out-edges go into the same node:



Canonicity of BDDs

- Reduced Ordered BDDs (RO-BDDs) [Bryant86]
  - Apply (algebraic) reduction rule until convergence
  - Nodes are ordered with respect to fixed variable order (ordered)

- We restrict our discussion to RO-BDDs (and just call them BDDs)

Theorem Fix variable order, then (RO)BDD for Boolean function f is unique

Proof Obtained from unique minimal automata theorem

Observation: given a BDD it takes constant time to check whether a formula is

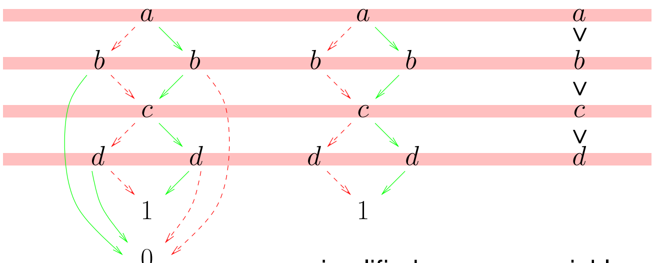
- a tautology,
- inconsistent,
- satisfiable

Aren't these hard?

Variable Ordering and BDD Size

Equality of two 2-bit vectors

$(a, c) = (b, d)$ equivalent to $(a \leftrightarrow b) \wedge (c \leftrightarrow d)$



(full) BDD

simplified visualization

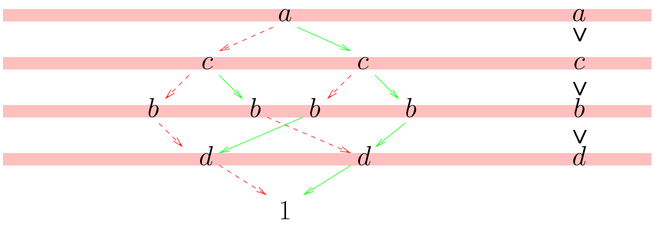
variable order

unique BDD for (one) optimal, **interleaved** variable ordering
(linear in the width of the bit vectors)

Variable Ordering and BDD Size

Equality of two 2-bit vectors

$(a, c) = (b, d)$ equivalent to $(a \leftrightarrow b) \wedge (c \leftrightarrow d)$



unique BDD for (one) worst case, **blocked** variable ordering
(exponential in the width of the bit vectors)

Selecting a Good Ordering

✘ Intractable problem

✘ Even when input is an 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

Data Structures for BDDs

- Nodes are numbered $0, 1, 2, 3, \ldots$
  (0, 1 are the terminals)

- Variables are numbered $1, 2, 3, \ldots, n$

Data Structures for BDDs

Node table:
$T : u \rightarrow (i, l, h)$

Operations:

- init(T)
- u := add(T, i, l, h)
- var(u)
- low(u)
- high(u)

Inverse of node table:
$H : (i, l, h) \rightarrow u$

Operations:

- init(H)
- u := lookup(H, i, l, h)
- insert(H, i, l, h, u)

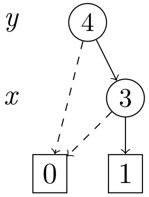
Node Table

Initial State:

#	var	low	$high$
0	$n+1$		
1	$n+1$		

(The $n+1$ variable number for 0/1 will become clear later).

Node Table: Example



#	*var*	*low*	*high*
0	3		
1	3		
2	–	–	–
3	2 x	0	1
4	1 y	0	3

Inserting a Node into T

```
MK[$T, H$]($i, l, h$)
    **if** $l = h$ **then**
        **return** $l$;
    **else if** lookup($H, i, l, h$)$\neq$ **then**
        **return** lookup($H, i, l, h$);
    **else**
        $u$ := add($T, i, l, h$);
        insert($H, i, l, h, u$);
        **return** $u$;
```

Building a BDD

Goal: build BDD for f.

Use Shannon's expansion as follows

$$f = (\neg x \wedge f|_{x=0}) \vee (x \wedge f|_{x=1})$$

to break problem into two subproblems. Solve subproblems recursively.

Building a BDD

```
BUILD[$T$,$H$]($f$)
    return BUILD2($f$, $1$);

function BUILD2($f$, $var$) =
    if $var > n$ then
        if $f$ is false then return $0$ else return $1$;
    else
        $f_0$ := BUILD2($f[0/x_i]$, $var + 1$);
        $f_1$ := BUILD2($f[1/x_i]$, $var + 1$);
        return MK[$T$,$H$]($var$, $f_0$, $f_1$);
end
```

Basic Operations on BDDs

- Canonicity implies
  - $f \equiv g$ iff. BDDs for f and g are equal
  - f tautology iff. BDD for f is 1
  - f satisfiable iff. BDD for f is not 0

- Basic operation: RESTRICT
  - $f|_{x=0}$: replace all x nodes by their low-edge sub-tree.
  - $f|_{x=1}$: replace all x nodes by their high-edge sub-tree.

Applying a Function

Conjunction, ...: $f(a,b,c,d) \star g(a,b,c,d)$,
where the symbol $\star$ denotes some binary operator.

Again, use Shannon's expansion as follows

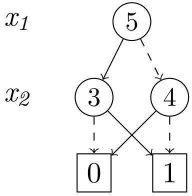
$$f \star g = \neg x \wedge (f|_{x=0} \star g|_{x=0}) \;\vee\; x \wedge (f|_{x=1} \star g|_{x=1})$$

to break problem into two subproblems. Solve subproblems recursively.

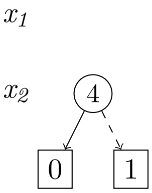
Applying a Function

```
function APPLY($u_1$, $u_2$) =
    if $u_1, u_2 \in \{0, 1\}$ then
        $u$ := $u_1 \star u_2$;
    else if var($u_1$) $=$ var($u_2$) then
        $u$ := MK(var($u_1$), APPLY(low($u_1$),low($u_2$)),
                              APPLY(high($u_1$),high($u_2$)));
    else if var($u_1$) $<$ var($u_2$) then
        $u$ := MK(var($u_1$), APP(low($u_1$),$u_2$), APP(high($u_1$),$u_2$));
    else      (* var($u_1$) $>$ var($u_2$) *)
        $u$ := MK(var($u_2$), APP($u_1$,low($u_2$)), APP($u_1$,high($u_2$)));
    return $u$;
```

Example

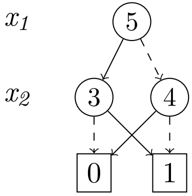


$x_1 \iff x_2$

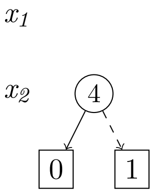


$\neg x_2$

Example



$x_1 \iff x_2$



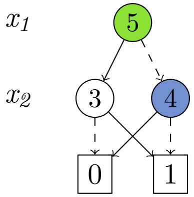
$\neg x_2$

Now compute BDD for $(x_1 \iff x_2) \vee \neg x_2$ using APPLY!

Example (cont.)

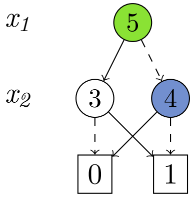
APPLY$(5, 4)$:

var(5)=1, var(4)=2



Example (cont.)

```
APPLY(5, 4):
    var(5)=1, var(4)=2
    MK(var(5), APP(low(5),4), APP(high(5),4));
```



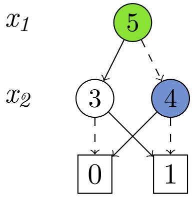
Example (cont.)

APPLY(5, 4):

var(5)=1, var(4)=2

MK(var(5), APP(low(5),4), APP(high(5),4));

MK(1, APP(4,4), APP(3,4));



Example (cont.)

APPLY$(5, 4)$:

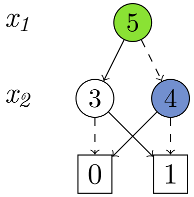
    var(5)=1, var(4)=2

    MK(var(5), APP(low(5),4), APP(high(5),4));

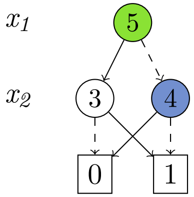
    MK(1, APP(4,4), APP(3,4));

    APP(4, 4):

        var(4)=var(4)=2



Example (cont.)



APPLY$(5, 4)$:

  var(5)=1, var(4)=2

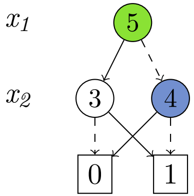
  MK(var(5), APP(low(5),4), APP(high(5),4));

  MK(1, APP(4,4), APP(3,4));

  APP$(4, 4)$:

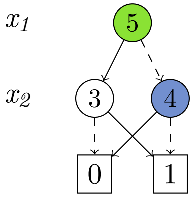
    var(4)=var(4)=2
    MK(2, APP(1,1), APP(0,0));

Example (cont.)



```
APPLY(5, 4):
    var(5)=1, var(4)=2
    MK(var(5), APP(low(5),4), APP(high(5),4));
    MK(1, APP(4,4), APP(3,4));
    APP(4, 4):
        var(4)=var(4)=2
        MK(2, APP(1,1), APP(0,0));
        MK(2, 1, 0);
```

Example (cont.)



APPLY(5, 4):

var(5)=1, var(4)=2

MK(var(5), APP(low(5),4), APP(high(5),4));

MK(1, APP(4,4), APP(3,4));

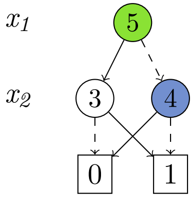
APP(4, 4):

var(4)=var(4)=2
MK(2, APP(1,1), APP(0,0));
MK(2, 1, 0); =4!

APP(3, 4):

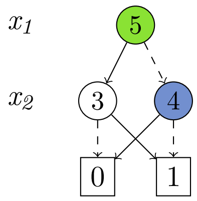
var(3)=var(4)=2

Example (cont.)



```
APPLY(5, 4):
    var(5)=1, var(4)=2
    MK(var(5), APP(low(5),4), APP(high(5),4));
    MK(1, APP(4,4), APP(3,4));
    APP(4, 4):
        var(4)=var(4)=2
        MK(2, APP(1,1), APP(0,0));
        MK(2, 1, 0); =4!
    APP(3, 4):
        var(3)=var(4)=2
        MK(2, APP(0,1), APP(1,0));
```

Example (cont.)



APPLY$(5, 4)$:

var(5)=1, var(4)=2

MK(var(5), APP(low(5),4), APP(high(5),4));

MK(1, APP(4,4), APP(3,4));

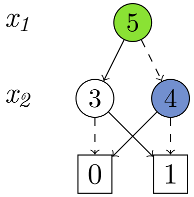
APP$(4, 4)$:

var(4)=var(4)=2
MK(2, APP(1,1), APP(0,0));
MK(2, 1, 0); $=4!$

APP$(3, 4)$:

var(3)=var(4)=2
MK(2, APP(0,1), APP(1,0));
MK(2, 1, 1);

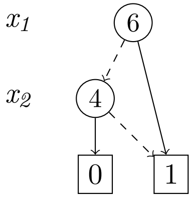
Example (cont.)



```
APPLY(5, 4):
    var(5)=1, var(4)=2
    MK(var(5), APP(low(5),4), APP(high(5),4));
    MK(1, APP(4,4), APP(3,4)); MK(1, 4, 1);
    APP(4, 4):
        var(4)=var(4)=2
        MK(2, APP(1,1), APP(0,0));
        MK(2, 1, 0); =4!
    APP(3, 4):
        var(3)=var(4)=2
        MK(2, APP(0,1), APP(1,0));
        MK(2, 1, 1); =1!
```

Example (cont.)

Final result:



#	*var*	*low*	*high*
0	3		
1	3		
2	–	–	–
3	–	–	–
4	2 x_2	1	0
5	–	–	–
6	1 x_1	4	1

Existential Quantification

For BDD f and a variable x compute BDD for

$$\exists\, x.\, f$$

Existential Quantification

For BDD f and a variable x compute BDD for

$$\exists\, x.\, f$$

This is equivalent to

$$f|_{x=0} \;\vee\; f|_{x=1}$$

- Two RESTRICT operations
- One call to APPLY

Summary: Binary Decision Diagrams (BDDs)

- Canonical representation of boolean functions
  - Every Boolean function has a unique RO-BDD
  - Try to obtain small graphs by means of sharing

✔ Efficient manipulation algorithms
(e.g., conjunction, quantification, . . .)

✘ Often explode in size