Introduction to Binary Decision Diagrams (BDDs)

Paul Jackson[1][2]
Paul.Jackson@ed.ac.uk

University of Edinburgh

Formal Verification
Autumn 2023

[1] Diagrams from Huth & Ryan, LiCS, 2nd Ed.
[2] Including contributions by Jacques Fleuriot and Bob Atkey

Model Checking needs Very Large Sets

Given a model $\mathcal{M} = \langle S, S_0, \rightarrow, L \rangle$ and a formula ϕ, the CTL model checking algorithm translates CTL formulas into sets of states:

$$[\![\phi]\!] \subseteq S$$

For realistic models, the size of S can be enormous.

Example: The NuSMV 2.6 distribution contains an example `guidance`, which is a model of part of the NASA Space Shuttle's autopilot. According to NuSMV:

```
NuSMV > print_reachable_states
############################################################
system diameter: 70
reachable states: 2.10443e+14 (2^47.5804) out of 2.63684e+27 (2^91.0909)
############################################################
```

If each state is represented using 96 bits, it would need at least approx 2.52 petabytes to explicitly store the set of all reachable states.

Boolean functions

- Notation: will use
  - 0,1 for $\bot$, $\top$
  - $+, \cdot, \bar{}$ for $\vee$, $\wedge$, $\neg$
- A Boolean function of n args is a function $\{0,1\}^n \to \{0,1\}$
  - Example: $f(x,y,z) \doteq x + y \cdot \bar{z}$
- As models are finite, we can use $\{0,1\}^k$ for set of states S
- Can represent a state subset $X \subseteq S$ using a Boolean function

  $f_X \in \{0,1\}^k \to \{0,1\}$ such that $f_X(s) = 1$ iff $s \in X$
- Can represent a binary relation on states (e.g. a transition relation) using a Boolean function.

  $g_\to \in \{0,1\}^{2k} \to \{0,1\}$ such that $g_\to(s,s') = 1$ iff $s \to s'$
- Operations on Boolean functions form basis of many model checking algorithms

Representations of Boolean functions

From H&R, Figure 6.1

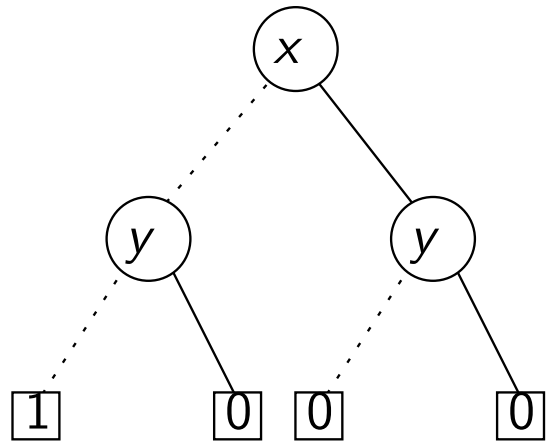
Representation of		test for		Boolean operations		
Boolean functions	compact?	satisf'ty	validity	$\cdot$	$+$	$^-$
Prop. formulas	often	hard	hard	easy	easy	easy
Formulas in DNF	sometimes	easy	hard	hard	easy	hard
Formulas in CNF	sometimes	hard	easy	easy	hard	hard
Truth tables	never	hard	hard	hard	hard	hard
Reduced OBDDs	often	easy	easy	medium	medium	easy

often/sometimes/never are indications of space complexity
hard/medium/easy are indications of time complexity

Note: With a truth table representation, while operations are conceptually easy, especially when table rows are always listed in some standard order, the time complexities are hard, as table sizes and hence operation time complexities are always exponential in the number of input variables.

Binary decision trees

Tree for Boolean function $f(x, y) \doteq \bar{x} \cdot \bar{y}$



To compute value

- Start at root
- Take dashed line if value of var at current node is 0
- Take solid line if value of var at current node is 1
- Function value is value at terminal node reached

Binary decision diagram

Similar to Binary Decision Tree, except that can nodes can have multiple in-edges.

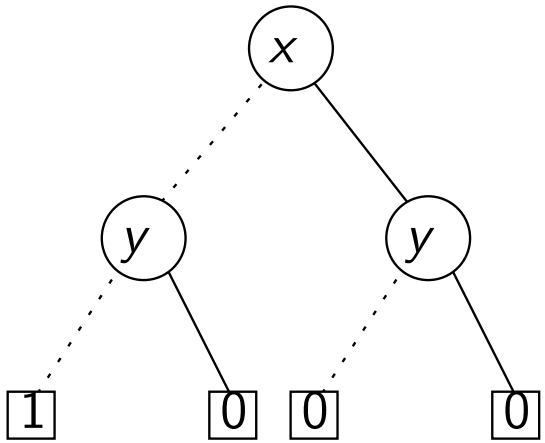
A binary decision diagram (BDD) is a finite DAG (Directed Acyclic Graph) with

- unique initial node,
- all non-terminals labelled with a Boolean variable,
- all terminals labelled with 0 or 1,
- all edges labelled with 0 (dashed edge) or 1 (solid edge),
- each non-terminal has exactly 1 out-edge labelled 0 and 1 out-edge labelled 1.

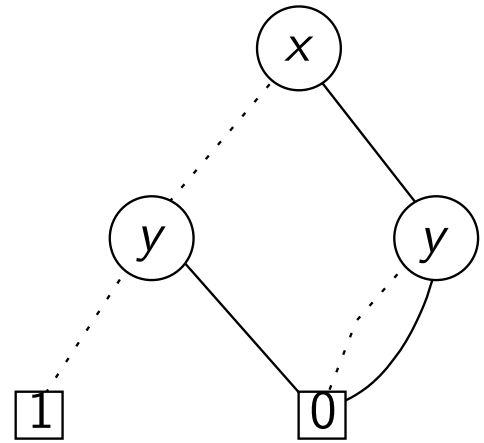
We will use BDDs with two extra properties

- *Reduced* - redundancy is eliminated
- *Ordered* - variables always occur in a given order

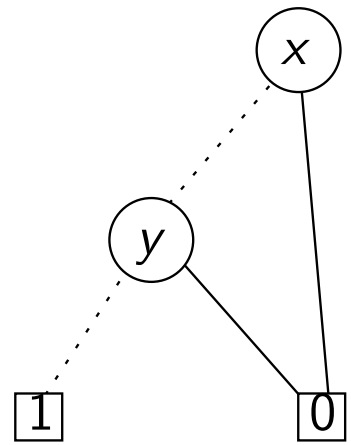
Reducing BDDs I



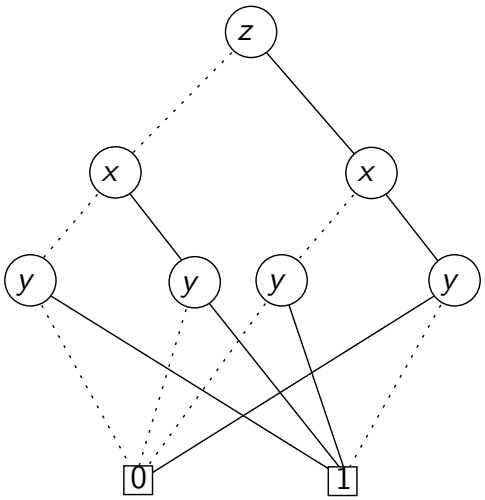
remove
duplicate
terminals



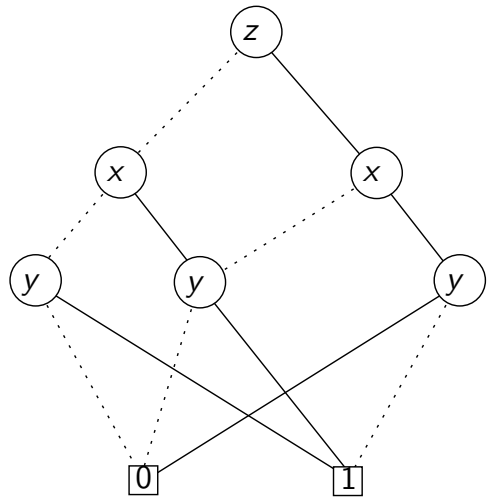
remove
redundant
test



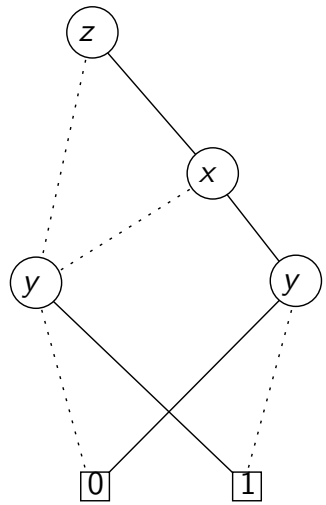
Reducing BDDs II



remove
duplicate
non-terminal



remove
duplicate
non-terminal
and
redundant
test

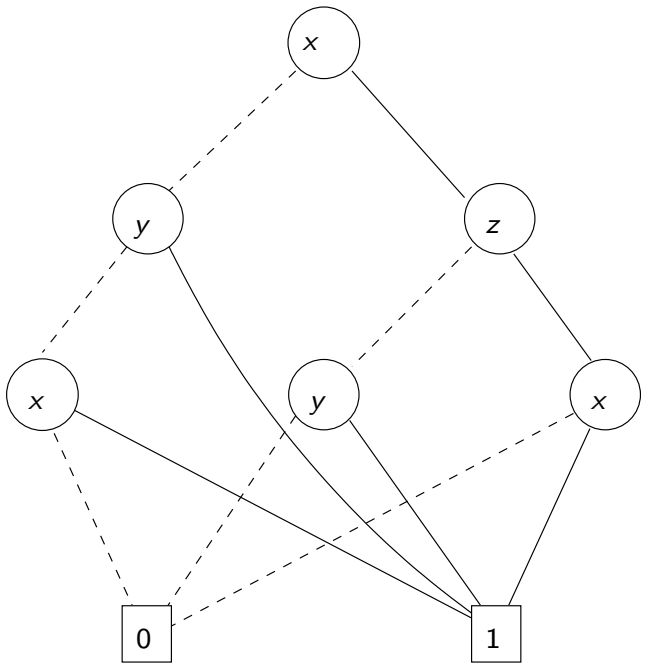


Reduction operations

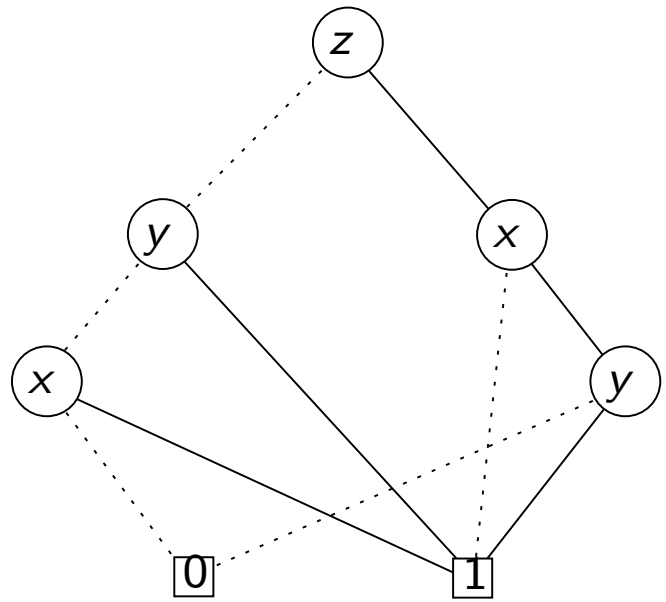
1. Removal of duplicate terminals. If a BDD contains more than one terminal 0-node, then redirect all edges which point to such a 0-node to just one of them. Proceed in the same way with terminal nodes labelled with 1.
2. Removal of redundant tests. If both outgoing edges of a node n point to the same node m, then eliminate that node n, sending all its incoming edges to m
3. Removal of duplicate non-terminals. If two distinct nodes n and m in the BDD are the roots of structurally identical subBDDS, then eliminate one of them, say m, and redirect all its incoming edges to the other one.

A BDD is reduced if it has been simplified as much as possible using these reduction operations

Generality of BDDs



A variable might occur more than once on a path



Ordering of variables on paths is not fixed

Ordered BDDs

- Let $[x_1, \ldots, x_n]$ be an ordered list of variables without duplicates
- A BDD B has an ordering $[x_1, \ldots, x_n]$ if
  - All variable labels of B occur in $[x_1, \ldots, x_n]$, and
  - if x_j follows x_i on a path in B, then $j > i$.
- An ordered BDD (OBDD) is a BDD which has an ordering for some list of variables
- The orderings of 2 OBDDs B and B' are compatible if there are no variables x, y such that
  - x is before y in the ordering for B, and
  - y is before x in the ordering for B'.

Theorem: *The reduced OBDD (ROBDD) representing a given function f is unique*. i.e.

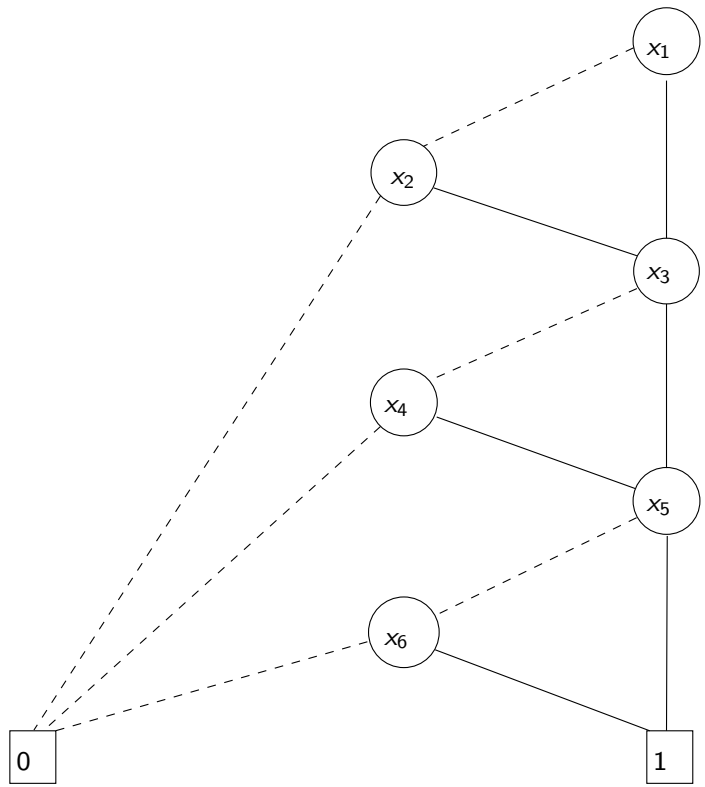
> If B and B' are two ROBDDs with compatible variable orderings representing the same Boolean function, then they have identical structure. (H&R Theorem 6.7)

Impact of variable ordering on size I

Consider the Boolean function
$(x_1 + x_2) \cdot (x_3 + x_4) \cdot \ \cdots \ \cdot (x_{2n-1} + x_{2n})$.

With variable ordering $[x_1, x_2, x_3, x_4, \ldots]$ ROBDD has $2n + 2$ nodes.

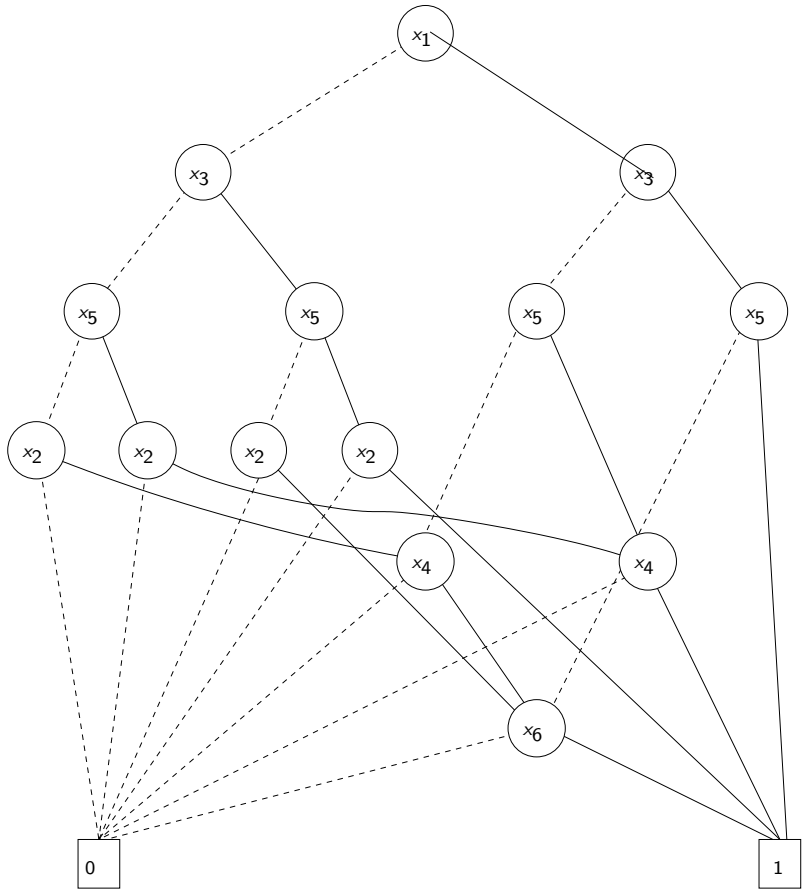
For $n = 3$:



Impact of variable ordering on size II

With ordering $[x_1, x_3, \ldots, x_{2n-1}, x_2, x_4, \ldots x_{2n}]$, size is 2^{n+1}.

For $n = 3$:

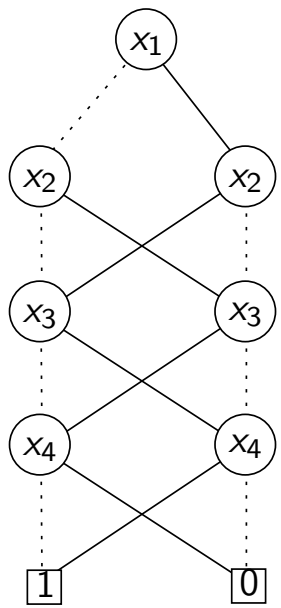


Exist heuristics for determining orderings that often work well in practice

Impact of variable ordering on size III

- Common ALU operations such as
  - shifts,
  - add and subtract,
  - bitwise and, or, exclusive or,
  - parity (whether a word has an odd or even number of 1s),

  all expressible using ROBDDs with total number of nodes linear in word size
- E.g. for even number of 1s for $n = 4$



- No efficient ROBDD representation for multiply operation

Importance of canonical representation

(canonical = unique, computable)

Having a canonical representation enables easy tests for

- Whether a variable is redundant. A Boolean function f does not depend on an input variable x if no nodes occur for x in the ROBDD for f.
- Semantic equivalence. Check if $f \equiv g$ by seeing if ROBDDs for f and g have identical structure
- Validity. Check if ROBDD is single terminal node $\boxed{1}$
- Satisfiability. Check if ROBDD is not the single terminal node $\boxed{0}$
- Implication. Check if $\forall \vec{x}.f(\vec{x}) \rightarrow g(\vec{x})$ by seeing if ROBDD for $f \cdot \bar{g}$ is $\boxed{0}$