Automated Reasoning

Lecture 18: Introduction to Binary Decision Diagrams (BDDs)

Robert Atkey
bob.atkey@ed.ac.uk

based on originals by Paul Jackson
diagrams from Huth & Ryan, LiCS, 2nd Ed.

Friday 20th March 2015

Recap

- Previously:
  - CTL and LTL Model Checking algorithms
- This time:
  - Binary Decision Diagrams
  - Reduced Binary Decision Diagrams
  - Reduced Ordered Binary Decision Diagrams

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.

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 2.5.4 distribution contains an example `guidance`, which is a model of (I think) part of the 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.

Representing states as Boolean functions

Idea: represent sets of states as boolean functions.

1. Represent each state as a binary string in $\{0,1\}^k$
2. Represent a set of states as a function $f\colon \{0,1\}^k \to \{0,1\}$

   $f(w) = 1$ if the state represented by w is in the set

   $f(w) = 0$ if the state represented by w is not in the set

   $\Rightarrow$ representation of sets by their **characteristic functions**

Representing states as Boolean functions

Idea: represent sets of states as boolean functions.

1. Represent each state as a binary string in $\{0,1\}^k$
2. Represent a set of states as a function $f\colon \{0,1\}^k \to \{0,1\}$
   $f(w) = 1$ if the state represented by w is in the set
   $f(w) = 0$ if the state represented by w is not in the set
   $\Rightarrow$ representation of sets by their **characteristic functions**

The set of all states could be represented as:

1. a data structure with 2.63684×10^{27} nodes; or
2. the boolean function:

$$f(w) = 1$$

How to represent boolean functions?

Representations of Boolean Functions

From H&R, Figure 6.1

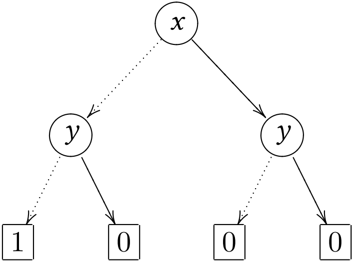
		test for		Operations		
Representation	compact?	satisf'y	validity	$\wedge$	$\vee$	$\neg$
Prop. Formulas	often	hard	hard	easy	easy	easy
Formulas in DNF	sometimes	easy	hard	hard	easy	hard
Formulas in CNF	sometimes	hard	easy	easy	hard	hard
Truth Tables	never	hard	hard	hard	hard	hard
Reduced OBDDs	often	easy	easy	medium	medium	easy

Easy/medium/hard refers to the time (or space) complexity in each case.

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 the boolean function $f(x, y) \doteq \neg x \wedge \neg y$



To compute value

1. Start at root
2. Take dashed line if value of var at current node is 0
3. Take solid line if value of var at current node is 1
4. Function value is value at terminal node reached

Binary decision diagram

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

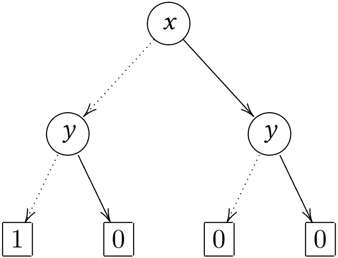
A **binary decision diagram** (BDD) is:
a finite DAG (Directed Acyclic Graph) with:

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

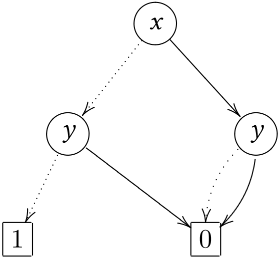
We will use BDDs with two extra properties:

1. *Reduced* – eliminate redundancy
2. *Ordered* – canonical ordering of the boolean variables

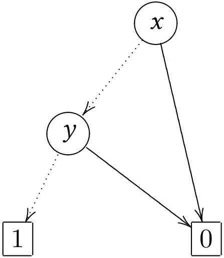
Reducing BDDs I



remove duplicate nontermi-nals $\longrightarrow$

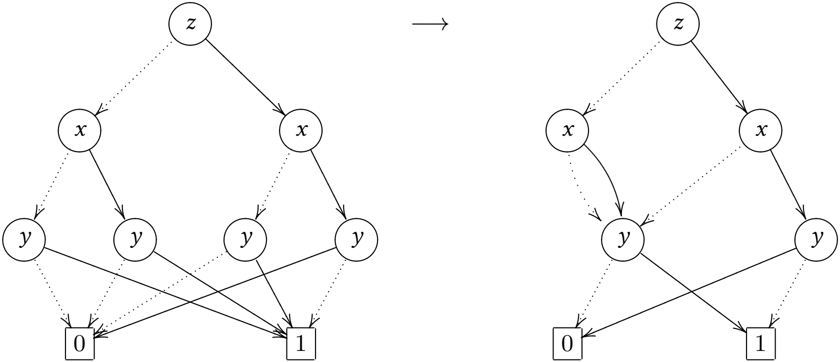


remove redundant test $\longrightarrow$



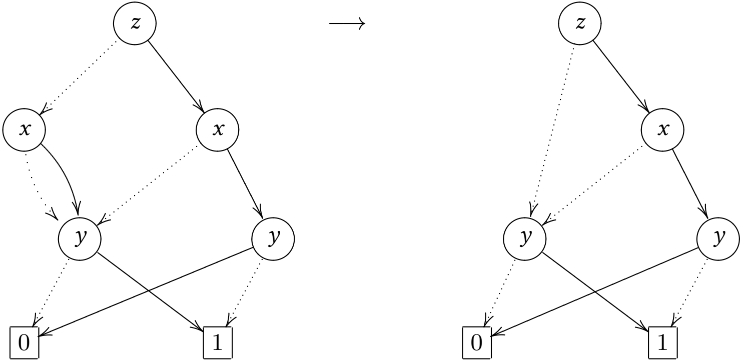
Reducing BDDs II

Removing duplicate non-terminals:



Reducing BDDs III

Removing redundant test:



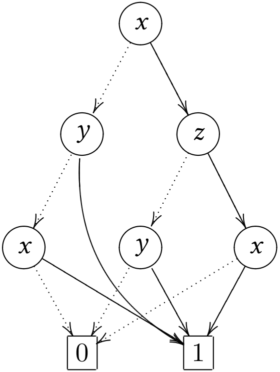
Reduction Operations

1. Removal of duplicate non-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. Do the same with terminal nodes labelled 1.
2. Removal of redundant tests. If both outgoing edges of a node n point to the same node m, then remove 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 and redirect all its incoming edges to the other one.

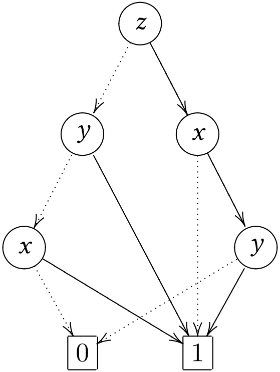
All of these operations preserve the BDD-ness of the DAG.

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

Generality of BDDs



Multiple occurrences of x



Different orderings on different paths

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
  1. all variables of B occur in $[x_1, \ldots, x_n]$; and
  2. 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 two OBBDs 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

*For a given ordering, the reduced OBDD (ROBDD) representing a given function f is **unique**.*

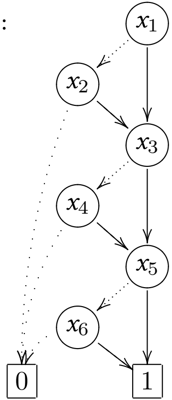
If B_1 and B_2 are two ROBDDs with compatible variable orderings representing the same boolean function, then they have identical structure.

Impact of variable ordering on size (I)

Consider the boolean function
$f(x_1, ..., x_{2n}) = (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge ... \wedge (x_{2n-1} \vee x_{2n})$

With variable ordering $[x_1, x_2, x_3, ..., x_{2n}]$ ROBDD has $2n + 2$ nodes

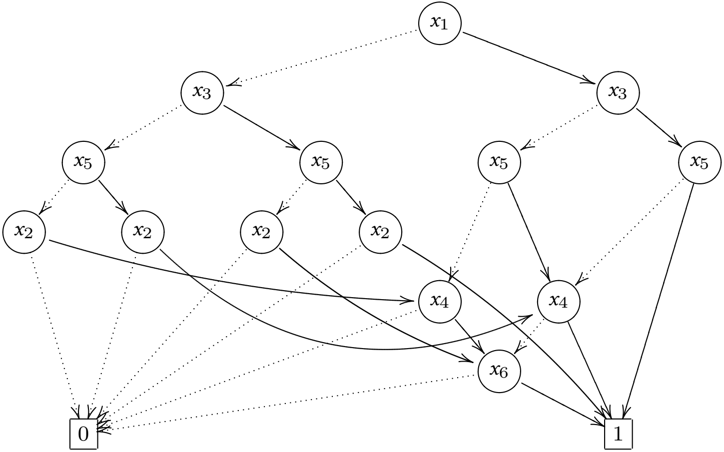
For $n = 3$:



Impact of variable ordering on size (II)

With $[x_1, x_3, \ldots, x_{2n-1}, x_2, x_4 \ldots, x_{2n}]$ ROBDD has 2^{n+1} nodes

For $n = 3$:



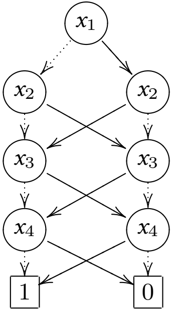
There are various heuristics that can help with choosing orderings.

However, improving a given ordering is NP-complete.

Impact of variable ordering on size III

Common ALU (Arithmetic Logic Unit) operations such as shifts, addition, subtraction, bitwise “and”, “or”, “exclusive or”, and xparity are 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 (they are all exponential size in the number of boolean variables).

Importance of canonical representation

Having a canonical representation enables easy tests for

- Absence of redundant variables. 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 $f \equiv g$ by checking whether or not the ROBDDs for f and g have identical structure.
- Validity. Check if the BDD is identical to the one with just the terminal node $\boxed{1}$ and nothing else.
- Satisfiability. Check if the BDD is not identical to the one with just the terminal node $\boxed{0}$ and nothing else.
- Implication. Check if $\forall \overrightarrow{x}.\, f(\overrightarrow{x}) \to g(\overrightarrow{x})$ by checking whether or not the ROBDD for $f \wedge \neg g$ is constant 0.

Memoisation

The representation of multiple ROBDDs can share memory.

Represent multiple BDDs using a large array of records:

v_0	l_0	h_0
v_1	l_1	h_1
v_2	l_2	h_2
v_3	l_3	h_3
v_4	l_4	h_4
...		
v_{n-1}	l_{n-1}	h_{n-1}

- each entry represents a BDD node
- v_i is the variable label;
- l_i is the index of the node pointed to for the false edge;
- h_i is the index of the node pointed to for the true edge;
- Use fake -1 and -2 indexes to represent $\boxed{0}$ and $\boxed{1}$.
- Each ROBDD is represented by the index of its root.

Use a lookup table to ensure each entry is unique. So identical ROBBDs (and hence semantically equal functions) will have exactly the same index.

Summary

- BDDs (H&R 6.1)
  - Why BDDs?
  - Binary Decision Diagrams
  - Reduced Binary Decision Diagrams
  - Reduced Ordered Binary Decision Diagrams
- Next time:
  - Algorithms for implementing logical operations on BDDs
  - More details on implementing CTL MC with BDDs