CS615: Formal Specification and Verification of Programs 2019

Lecture 18: Hardware Model Checking - BDD

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2019-11-05

Efficient symbolic operators

A symbolic model checker needs the following operations.

- $\vee$
- $\Rightarrow$ checker
- sp

In hardware, we have only Boolean variables. Therefore, a finite state space.

In hardware verification, we may have effective operations for the above.

Topic 18.1

Hardware model checking — Binary Decision Diagrams

Symbolic model checking and Hardware verification

Symbolic model checking for hardware using BDD Since hardware has only Boolean variables

First practical model checker

Binary Decision Diagram(BDD) is a data structure that enabled the first practical model checker.

BDDs came to prominence in early 90s.

J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang.
Symbolic model checking: 10^{20} states and beyond. Information and Computation, 1992.

BDD is no more the best known method. But, it is worth exploring.

Partial evaluation

Let us suppose a partial model m s.t. $\mathbf{Vars}(F) \not\subseteq dom(m)$.
We can assign meaning to $m(F)$, which we will denote with $F|_m$.

Definition 18.1

Let F be a formula and $m = \{p_1 \mapsto b_1, ..\}$ be a partial model.

$$\text{Let } F|_{x_i \mapsto b_i} \triangleq \begin{cases} F[\top/x_i] & \text{if } b_i = 1 \\ F[\bot/x_i] & \text{if } b_i = 0. \end{cases}$$

The partial evaluation $F|_m$ be $F|_{p_1 \mapsto b_1}|_{p_2 \mapsto b_2}| \ldots$ after some simplifications.

For short hand, we may write $F|_p$ for $F|_{p \mapsto 1}$ and $F|_{\neg p}$ for $F|_{p \mapsto 0}$.

Exercise 18.1

Prove $(F|_p \wedge p) \vee (F|_{\neg p} \wedge \neg p) \equiv F$

Commentary: We have seen similar definition the proof complexity lecture.

Example : partial evaluation

Example 18.1

Consider $F = (p \vee q) \wedge r$

$$F|_p = ((p \vee q) \wedge r)[\top/p] = (\top \vee q) \wedge r \equiv \top \wedge r \equiv r$$

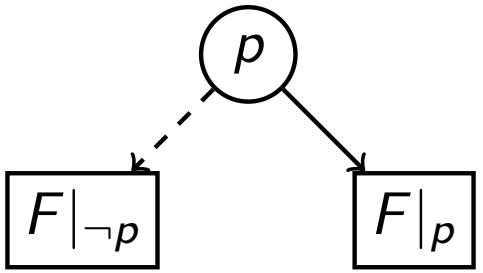
Exercise 18.2

Compute

- $((p \vee q) \wedge r)|_{\neg p}$
- $((p_1 \Leftrightarrow q_1) \wedge (p_2 \Leftrightarrow q_2))|_{p_1 \mapsto 0, p_2 \mapsto 0}$

Decision branch

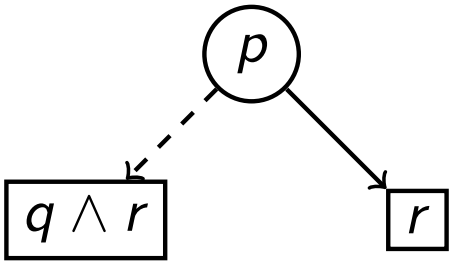
Due to the theorem in exercise 18.1, the following tree may be viewed as representing F.



Dashed arrows represent 0 decisions and solid arrows represent 1 decisions.

Example 18.2

Consider $(p \vee q) \wedge r$

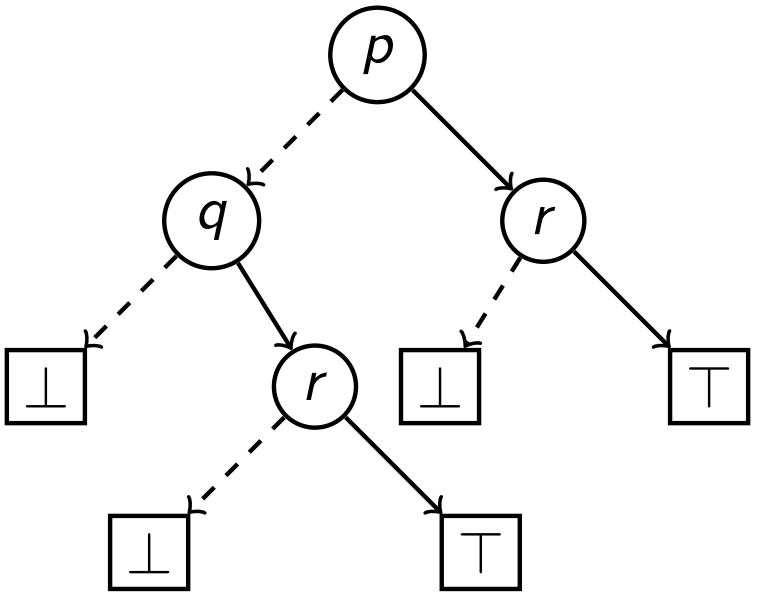


Decision tree

We may further expand $F|_{\neg p}$ and $F|_p$ until we are left with $\top$ and $\bot$ at the leaves. The obtained tree is called the decision tree for F.

Example 18.3

Consider $(p \vee q) \wedge r$



Binary decision diagram(BDD)

If two nodes represent same formula, we may rewire the incoming edges to only one of the nodes.

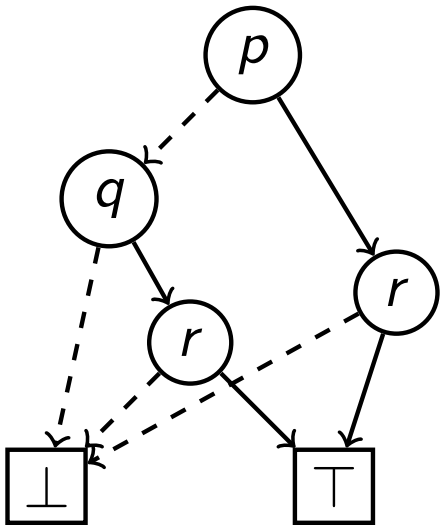
Definition 18.2

A BDD is a finite DAG such that

- *each internal node is labeled with a propositional variable*
- *each internal node has a low (dashed) and a high child (solid)*
- *there are exactly two leaves one is labelled with $\top$ and the other with $\bot$*

Example 18.4

The following is a BDD for $(p \vee q) \wedge r$



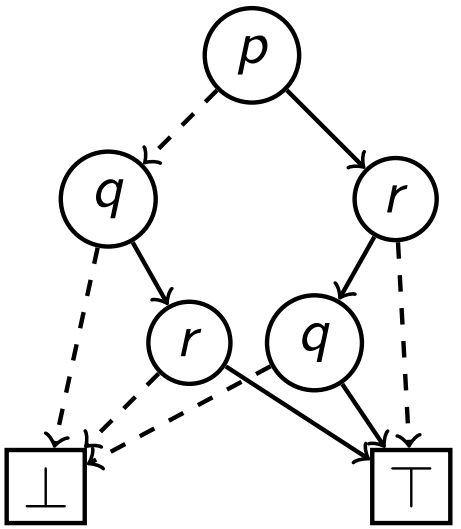
Ordered BDD (OBDD)

Definition 18.3

A BDD is ordered if there is an order $<$ over variables including $\top$ and $\bot$ s.t. for each node v, $v < low(v)$ and $v < high(v)$.

Example 18.5

The following BDD is not an ordered BDD



Exercise 18.3

a. *Convert the above BDD into a formula*

b. *Give an ordered BDD of the formula*

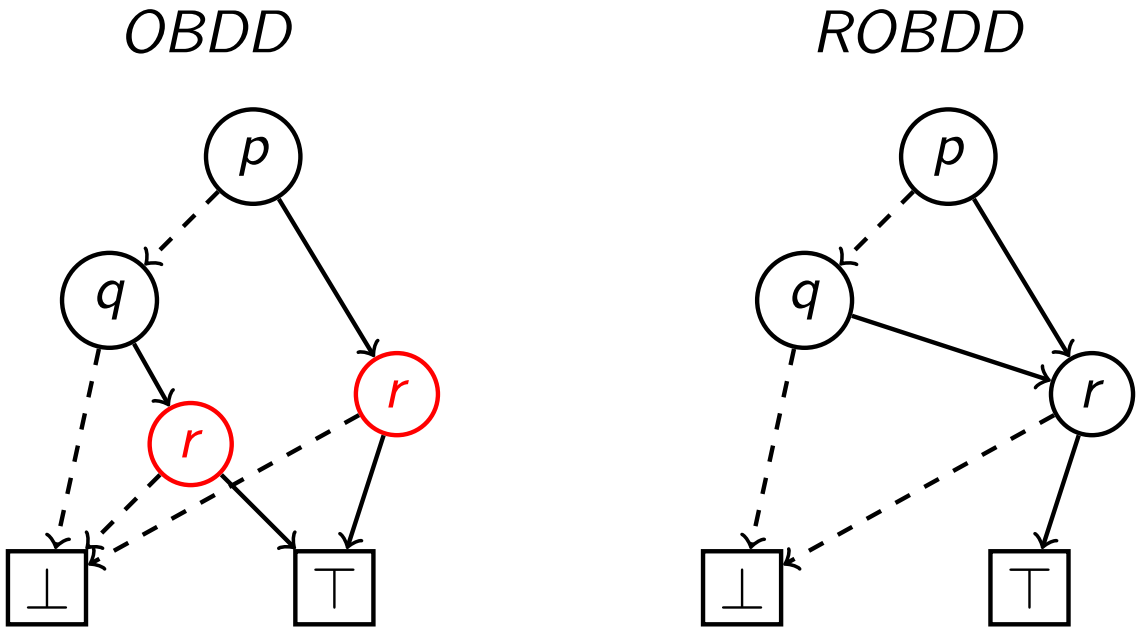
Reduced OBDD (ROBDD)

Definition 18.4

A OBDD is reduced if

- *for any nodes u and v, if $var(u) = var(v)$, $low(u) = low(v)$, $high(u) = high(v)$ then $u = v$*
- *for each node u, $low(u) \neq high(u)$*

Example 18.6



Converting to ROBDD

Any OBDD can be converted into ROBDD by iteratively applying the following transformations.

1. If there are nodes u and v such that $var(u) = var(v)$, $low(u) = low(v)$, $high(u) = high(v)$ then remove u and connect all the parents of u to v.

2. If there is a node u such that $low(u) = high(u)$ then remove u and connect all the parents of u to $low(u)$.

Exercise 18.4

Prove the above iterations terminate.

Canonical ROBDD

Theorem 18.1

For a function $f : \mathcal{B}^n \to \mathcal{B}$ there is exactly one ROBDD u with ordering $p_1 < \cdots < p_n$ such that u represents $f(p_1, \ldots, p_n)$.

Proof.

We use the induction over the number of parameters.

base (n=0): There are only two functions $f() = 0$ and $f() = 1$, which are represented by nodes $\bot$ and $\top$ respectively.

step ($n > 0$): Assume, unique ROBDD for functions with n parameters. Consider a function $f : \mathcal{B}^{n+1} \to \mathcal{B}$.

Let $f_0(p_2, \ldots, p_{n+1}) = f(0, p_2, \ldots, p_{n+1})$ which is represented by ROBDD u_0.
Let $f_1(p_2, \ldots, p_{n+1}) = f(1, p_2, \ldots, p_{n+1})$ which is represented by ROBDD u_1.
...

Canonical ROBDD (cond.) II

Proof(contd.)

case $u_0 = u_1$:

Therefore, $f = f_0 = f_1$. Therefore, u_0 represents f.

Assume there is $u' \neq u_0$ that represents f.

Therefore, $var(u') = p_1$ (why?), $low(u') = high(u') = u_0$.

Therefore, u' is not a ROBDD. ...

Canonical ROBDD (cond.) III

Proof(contd.)

case $u_0 \neq u_1$:

Let u be such that $var(u) = p_1$, $low(u) = u_0$, and $high(u) = u_1$.

Clearly, u is a ROBDD.

Assume there is $u' \neq u$ that represents f. Therefore, $var(u') = p_1$ (why?).

Due to induction hyp., $low(u') = u_0$, and $high(u') = u_1$.

Due to the reduced property, $u = u'$. □

Satisfiablility via BDD

Build a ROBDD that represents F and unsat formulas have only one node $\perp$.

Benefits of ROBDD

- If intermediate ROBDDs are small then the satisfiability check will be efficient.
- Cost of computing ROBDDs vs sizes of BDDs
- Due to the canonicity property, ROBDD is used as a formula store
- Various operations on the ROBDDs are conducive to implementation

Issues with ROBDD

- BDDs are very sensitive to the variable ordering. There are formulas that have exponential size ROBDDs for some orderings
- There is no efficient way to detect good variable orderings

Exercise 18.5

Draw the ROBDD for

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$$

with the following ordering on variables $x_1 < x_3 < x_2 < x_4$.

Topic 18.2

Algorithms for BDDs

Algorithms for BDDs

Next we will present algorithms for BDDs to illustrate the convenience of the data structure.

Global data structures

The algorithms maintain the following two global data structures.

$store = (Nodes, low, high, var) := (\{\bot, \top\}, \lambda x.\texttt{null}, \lambda x.\texttt{null}, \lambda x.\texttt{null})$

$reverseMap : (\textbf{Vars} \times Nodes \times Nodes) \to Nodes := \lambda x.\texttt{null}$

Constructing a BDD node

Algorithm 18.1: MakeNode(p, u_0, u_1)

Input: $p \in$ **Vars**, $u_0, u_1 \in$ *Nodes*

if $u_0 = u_1$ **then return** u_0 ;

if *reverseMap.exists*(p, u_0, u_1) **then return**
reverseMap.lookup(p, u_0, u_1) ;

$u :=$ *store.add*(p, u_0, u_1); *reverseMap.add*$((p, u_0, u_1), u)$;

Constructing BDDs from a formula

Algorithm 18.2: $\textsc{BuildROBDD}(F, p_1 < \cdots < p_n)$

Input: $F(p_1, \ldots, p_n) \in \mathbf{P}$, $p_1 < \cdots < p_n$: an ordering over variables of F

if $n = 0$ **then**

  if $F \equiv \bot$ **then return** $\bot$; **else return** $\top$;

$u_0 := \textsc{BuildROBDD}(F|_{\neg p_1}, p_2 < \cdots < p_n)$;

$u_1 := \textsc{BuildROBDD}(F|_{p_1}, p_2 < \cdots < p_n)$;

return $\textsc{MakeNode}(p_1, u_0, u_1)$

Conjunction of BDDs

Algorithm 18.3: $\textsc{ConjBDDs}(u, v)$

Input: u and v ROBDDs with same variable ordering

if $u = \bot$ *or* $v = \top$ **then return** u;

if $u = \top$ *or* $v = \bot$ **then return** v;

$u_0 := low(u); u_1 := high(u); p_u := var(u)$;

$v_0 := low(v); v_1 := high(v); p_v := var(v)$;

if $p_u = p_v$ **then**

  return $\textsc{MakeNode}(p_u, \textsc{ConjBDDs}(u_0, v_0), \textsc{ConjBDDs}(u_1, v_1))$

if $p_u < p_v$ **then**

  return $\textsc{MakeNode}(p_u, \textsc{ConjBDDs}(u_0, v), \textsc{ConjBDDs}(u_1, v))$

if $p_u > p_v$ **then**

  return $\textsc{MakeNode}(p_u, \textsc{ConjBDDs}(u, v_0), \textsc{ConjBDDs}(u, v_1))$

Exercise 18.6

a. Write an algorithm for computing disjunction of BDDs

b. Write an algorithm for computing not of BDDs.

Restriction on a value

Algorithm 18.4: RESTRICT(u, p, b)

Input: u ROBDD with same variable ordering, variable p, $b \in \mathcal{B}$

$u_0 := low(u); u_1 := high(u); p_u := var(u);$

if $p_u = p$ *and* $b = 0$ **then**
  return RESTRICT(u_0, p, b)

if $p_u = p$ *and* $b = 1$ **then**
  return RESTRICT(u_1, p, b)

if $p_u < p$ **then**
  return MAKENODE$(p_u$,RESTRICT(u_0, p, b),RESTRICT$(u_1, p, b))$

if $p_u > p$ **then**
  return u

Impact of BDDs

- In 90s, BDDs revolutionized hardware verification
- Later other methods were found that are much faster and the fall of BDD was marked by the following paper,

  A. Biere,A. Cimatti,E. Clarke,Y. Zhu,
  Symbolic Model Checking without BDDs, TACAS 1999

- However, BDDs are still the heart of various software packages

Topic 18.3

Problems

Take and of the BDDs

Exercise 18.7

Construct ROBDD of the following formula for the order $p < q < r < s$.

$$F = (p \vee (q \oplus r) \vee (p \vee s))$$

Let u be the ROBDD node that represents F.
Give the output of $\textsc{Restrict}(u_F, p, b)$

Variable reordering

Exercise 18.8

Let u be an ROBDD with variable ordering $p_1 < ... < p_n$.
Give an algorithm to transforming u into a ROBDD with ordering
$p_1 < .. < p_{i-1} < p_{i+1} < p_i < p_{i+2} < .. < p_n$.

BDD-XOR

Exercise 18.9

Write an algorithm for computing xor of BDDs

BDD encoding

Exercise 18.10

Consider a and b be 2 bit wide bit-vectors. Write BDD of each of three output bits in the following bit-vector addition.

$$a + b$$

BDD model counting

Exercise 18.11

a. Give an algorithm for counting models for a given ROBDD.

b. Does this algorithm work for any BDD?

End of Lecture 18