Concurrent Object Oriented Languages

Binary Decision Diagrams

https://wiki.cse.yorku.ca/course/6490A

Model checking

- *Explicit*: states and transitions are represented explicitly.

  Drawback: the state space of interesting systems is usually too large to represent explicitly.
- *Symbolic*: (sets of) states and (sets of) transitions are represented symbolically.

  Key idea: exploit the fact that the state space of most systems is not random.

  We focus on one symbolic approach:
  - BDD based

Satisfiability

Cook's theorem

Satisfiability checking of Boolean expressions is NP-complete.

Stephen Cook

- recipient of the ACM Turing award (1982)
- fellow of the Royal Society of London (1998)
- fellow of the Royal Society of Canada (1984)
- member of the National Academy of Sciences (1985)
- member of the American Academy of Arts and Sciences (1986)

Source: Jiri Janicek

Tautology

Theorem

Tautology checking of Boolean expressions is co-NP-complete.

Disjunctive normal form

Definition

A *literal* is a variable or its negation.

Definition

A Boolean expression is in *disjunctive normal form (DNF)* if it is a disjunction of conjunctions of literals.

Proposition

Any Boolean expression is equivalent to one in DNF.

Proposition

Satisfiability checking of Boolean expressions in DNF is in P.

Proposition

Tautology checking of Boolean expressions in DNF is co-NP-complete.

Conjunctive normal form

Definition

A *clause* is a disjunction of literals.

Definition

A Boolean expression is in *conjunctive normal form (CNF)* if it is a conjunction of clauses.

Proposition

Any Boolean expression is equivalent to one in CNF.

Proposition

Satisfiability checking of Boolean expressions in CNF is NP-complete.

Proposition

Tautology checking of Boolean expressions in CNF is in P.

If-then-else normal form

Notation

$$\begin{array}{rcl} 0 & : & \text{false} \\ 1 & : & \text{true} \\ x \rightarrow t_1, t_0 & : & (x \wedge t_1) \vee (\neg x \wedge t_0) \end{array}$$

Definition

The set of Boolean expressions in *if-then-else normal form (INF)* is defined by

$$t ::= 0 \mid 1 \mid x \rightarrow t, t$$

If-then-else normal form

Question

Give a Boolean expression in INF equivalent to $x_1 \wedge (\neg x_2 \vee x_3)$.

If-then-else normal form

Question

Give a Boolean expression in INF equivalent to $x_1 \wedge (\neg x_2 \vee x_3)$.

Answer

$$
\begin{aligned}
t &= x_1 \rightarrow t_1, t_0 \\
t_0 &= x_2 \rightarrow t_{01}, t_{00} \\
t_1 &= x_2 \rightarrow t_{11}, t_{10} \\
t_{00} &= x_3 \rightarrow 0, 0 \\
t_{01} &= x_3 \rightarrow 0, 0 \\
t_{10} &= x_3 \rightarrow 1, 1 \\
t_{11} &= x_3 \rightarrow 1, 0
\end{aligned}
$$

If-then-else normal form

Shannon's expansion theorem

For every Boolean expression t and variable x,

$$t = x \to t[1/x], t[0/x].$$

Proposition

Any Boolean expression is equivalent to one in INF.

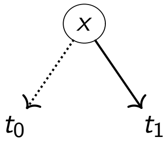
Decision trees

Boolean expressions in INF can be viewed as binary trees known as *decision trees*.

Two types of leaves: 0 and 1

$\boxed{0}$ $\boxed{1}$

One type of internal nodes: $x \longrightarrow t_1, t_0$



Question

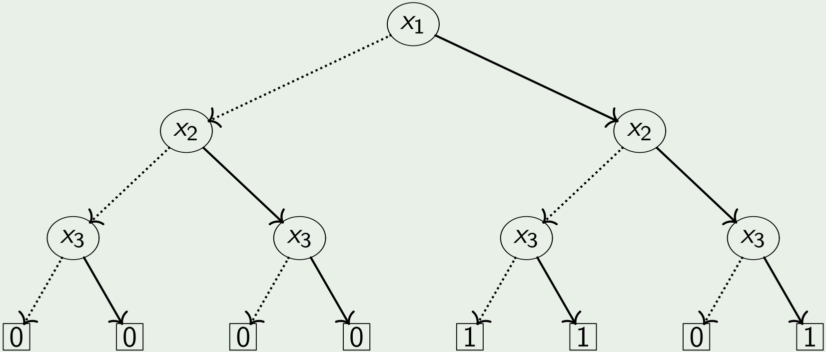
Draw the decision tree for the Boolean expression in INF equivalent to $x_1 \wedge (\neg x_2 \vee x_3)$.

Decision trees

Question

Draw the decision tree for the Boolean expression in INF equivalent to $x_1 \wedge (\neg x_2 \vee x_3)$.

Answer



If-then-else normal form

$$
\begin{aligned}
t &= x_1 \rightarrow t_1, t_0 \\
t_0 &= x_2 \rightarrow t_{01}, t_{00} \\
t_1 &= x_2 \rightarrow t_{11}, t_{10} \\
t_{00} &= x_3 \rightarrow 0, 0 \\
t_{01} &= x_3 \rightarrow 0, 0 \\
t_{10} &= x_3 \rightarrow 1, 1 \\
t_{11} &= x_3 \rightarrow 1, 0
\end{aligned}
$$

Question

Identify all equal subexpressions.

If-then-else normal form

$$
\begin{aligned}
t &= x_1 \rightarrow t_1, t_0 \\
t_0 &= x_2 \rightarrow t_{01}, t_{00} \\
t_1 &= x_2 \rightarrow t_{11}, t_{10} \\
t_{00} &= x_3 \rightarrow 0, 0 \\
t_{01} &= x_3 \rightarrow 0, 0 \\
t_{10} &= x_3 \rightarrow 1, 1 \\
t_{11} &= x_3 \rightarrow 1, 0
\end{aligned}
$$

Question

Identify all equal subexpressions.

Answer

There are multiple occurrences of 0 and 1. Furthermore, t_{00} and t_{01} are equal.

Binary decision diagram

Question

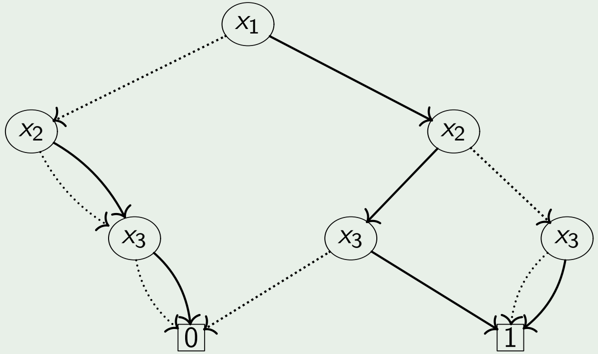
Identify the equal subtrees in the decision tree for the Boolean expression in INF equivalent to $x_1 \wedge (\neg x_2 \vee x_3)$.

Binary decision diagram

Question

Identify the equal subtrees in the decision tree for the Boolean expression in INF equivalent to $x_1 \wedge (\neg x_2 \vee x_3)$.

Answer



Binary decision diagram

Definition

A *binary decision diagram (BDD)* is a rooted directed acyclic graph where

- two (external) nodes where have out-degree zero and are labelled 0 and 1,
- and all other (internal) nodes have out-degree two, with one outgoing edge called the low edge and the other called the high edge, and are labelled with a variable.

Binary decision diagram

Definition

A *binary decision diagram (BDD)* is a rooted directed acyclic graph where

- two (external) nodes where have out-degree zero and are labelled 0 and 1,
- and all other (internal) nodes have out-degree two, with one outgoing edge called the low edge and the other called the high edge, and are labelled with a variable.

Notation

Let u be an internal node.
$var(u)$ denotes the variable with which node u is labelled.
$low(u)$ denotes the successor of node u along its low edge (corresponding to the case that value of $var(u)$ is low, that is, 0).
$high(u)$ denotes the successor of node u along its high edge (corresponding to the case that value of $var(u)$ is high, that is, 1).

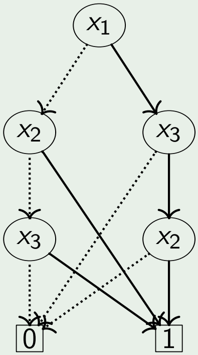
Ordered binary decision diagrams

Definition

A BDD is *ordered* if on all paths through the graph the variables respect a given linear order $x_1 < x_2 < \cdots < x_n$.

Question

Is the BDD



ordered?

Reduced ordered binary decision diagrams

Definition

An ordered BDD is *reduced* if

- *unique*: no two distinct internal nodes u and v have the same variable, low- and high-successor, that is,

  if $var(v) = var(u), low(v) = low(u),$ and $high(v) = high(u)$ then $u = v$.

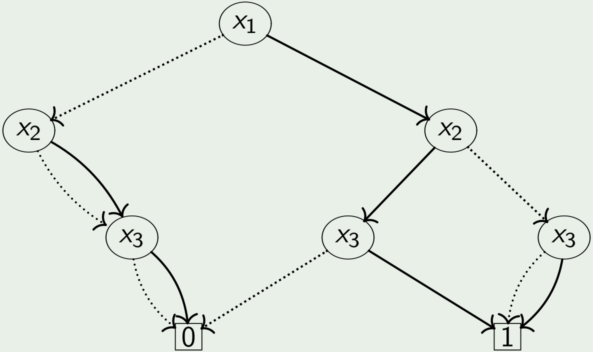
- *non-redundant*: no internal node u has identical low- and high-successor, that is,

$$low(u) \neq high(u).$$

Reduced ordered binary decision diagrams

Question

Is the ordered BDD



reduced?

Reduced ordered binary decision diagrams

Question

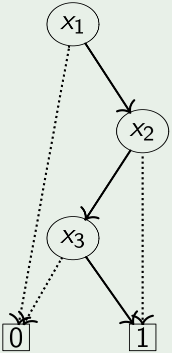
What is the corresponding reduced ordered BDD?

Reduced ordered binary decision diagrams

Question

What is the corresponding reduced ordered BDD?

Answer



Canonicity lemma

Lemma

For a Boolean expression t with variables $x_1, x_2, \ldots, x_n$ and a linear order $x_1 < x_2 < \cdots < x_n$, there exists a unique reduced ordered BDD which is equivalent to t.

For the remainder, we restrict our attention to reduced ordered BDDs and simply call them BDDs.

Randal Bryant

- member of the National Academy of Engineering (2003),
- recipient of the Paris Kanellakis Theory and Practice Award (1997)
- recipient of the IEEE Emanuel R. Piore Award (2007)
- his paper on BDDs is one of the most cited computer science papers (more than 8000 citations)

Source: Randal Bryant

Proposition

Satisfiability checking of BDDs is constant time.

Proposition

Tautology checking of BDDs is constant time.

The variable order matters

Question

Draw the BDD corresponding to

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6)$$

for the variable ordering

$$x_1 < x_2 < x_3 < x_4 < x_5 < x_6$$

The variable order matters

Question

Draw the BDD corresponding to

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6)$$

for the variable ordering

$$x_1 < x_4 < x_5 < x_2 < x_3 < x_6$$

The variable order matters

Theorem

Deciding whether a given variable order is optimal is NP-hard.

Heuristics are used to find good variable orderings. For more details, see, for example,
I. Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications.* 2000.

Data structures for BDDs

The nodes are represented as integers 0, 1, 2, . . . where 0 and 1 represent the leaves labelled 0 and 1.

Given a variable ordering $x_1 < x_2 < \cdots < x_n$, the variables are represented by their indices 0, 1, . . . , n.

Node table

The *node table* can be viewed as a partial function

$$T : \mathbb{N} \to (\mathbb{N}^3 \cup \mathbb{N})$$

which maps the index of a node to the indices of its variable, low- and high-successor.

$$u \mapsto (v, \ell, h)$$

Note that 0 and 1 do not have a low- and high-successor. These external vertices are assigned a variable index which is $n + 1$, where n is the number of variables. (This choice simplifies some of the algorithms to be discussed later.)

Operations on node table

init(T): initializes T to contain only nodes 0 and 1.

u	$var(u)$	$low(u)$	$high(u)$
0	$n+1$		
1	$n+1$		

Operations on node table

$u \leftarrow add(T, i, \ell, h)$: allocate a new node u with attributes (i, ℓ, h).

Question

Given the node table

u	$var(u)$	$low(u)$	$high(u)$
0	$n+1$		
1	$n+1$		

what does the operation $add(T, 4, 1, 0)$ return?

Operations on node table

$u \leftarrow add(T, i, \ell, h)$: allocate a new node u with attributes (i, ℓ, h).

Question

Given the node table

u	$var(u)$	$low(u)$	$high(u)$
0	$n+1$		
1	$n+1$		

what does the operation $add(T, 4, 1, 0)$ return?

Answer

2.

Operations on node table

$u \leftarrow add(T, i, \ell, h)$: allocate a new node u with attributes (i, ℓ, h).

Question

Given the operation $add(T, 4, 1, 0)$ applied to the node table

u	$var(u)$	$low(u)$	$high(u)$
0	5		
1	5		

what is the resulting node table?

Operations on node table

$u \leftarrow add(T, i, \ell, h)$: allocate a new node u with attributes (i, ℓ, h).

Question

Given the operation $add(T, 4, 1, 0)$ applied to the node table

u	$var(u)$	$low(u)$	$high(u)$
0	5		
1	5		

what is the resulting node table?

Answer

u	$var(u)$	$low(u)$	$high(u)$
0	5		
1	5		
2	4	1	0

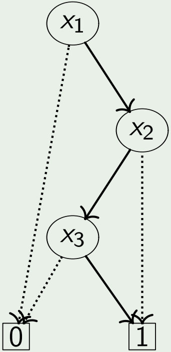
Operations on node table

$var(u)$	:	look up the var attribute of u in T
$low(u)$	:	look up the low attribute of u in T
$high(u)$	:	look up the high attribute of u in T

Example of node table

Question

Give the node table corresponding to the BDD



Example of node table

Answer

u	$var(u)$	$low(u)$	$high(u)$
0	4		
1	4		
2	3	0	1
3	2	1	2
4	1	0	3

Inverse of node table

The *inverse of the node table* can be viewed as a partial function

$$H : \mathbb{N}^3 \to \mathbb{N}$$

which maps the indices of the attributes of a node to the index of the node.

$$(v, \ell, h) \mapsto u$$

For all $u \geq 2$,

$$T(u) = (i, \ell, h) \text{ iff } H(i, \ell, h) = u.$$

Operations on inverse of node table

$$
\begin{aligned}
init(H) &\quad : \quad \text{initializes } H \text{ to be empty} \\
b \leftarrow member(H, i, \ell, h) &\quad : \quad \text{check if } (i, \ell, h) \text{ is in } H \\
u \leftarrow lookup(H, i, \ell, h) &\quad : \quad \text{find } H(i, \ell, h) \\
insert(H, i, \ell, h, u) &\quad : \quad \text{make } (i, \ell, h) \text{ map to } u \text{ in } H
\end{aligned}
$$

Operations on BDDs

Question

Consider the node table T and its inverse H.

- Let ℓ and h be indices of nodes u_ℓ and u_h.
- Let i be the index of variable x_i.[a]

Return the index of the node of T corresponding to $x_i \rightarrow u_h, u_\ell$ and expand T and H if needed.

[a] In the variable ordering, this variable occurs before all variables occurring in the subgraphs rooted at ℓ and h.

Operations on BDDs

$\textsc{Mk}[T, H](i, \ell, h)$

```
if ℓ = h then
   return ℓ
else if member(H, i, ℓ, h) then
   return lookup(H, i, ℓ, h)
else
   u ← add(T, i, ℓ, h)
   insert(H, i, ℓ, h)
   return u
```

Operations on BDDs

Question

Consider the node table T and its inverse H. Let t be a Boolean expression. Return the node of T corresponding to t.

```
BUILD[T, H](t)
   return build(t, 1)

   function build(t, i)
      if i > n then
         if t is false then return 0 else return 1
      else
         u0 ← (t[0/xi], i + 1)
         u1 ← (t[1/xi], i + 1)
         return MK(i, u0, u1)
```

Operations on BDDs

Proposition

For all Boolean binary operators $\otimes$,

$$(x \rightarrow t_1, t_0) \otimes (x \rightarrow u_1, u_0) = x \rightarrow t_1 \otimes u_1, t_0 \otimes u_0.$$

Operations on BDDs

Question

Consider the node table T and its inverse H.

- Let u_1 and u_2 be indices of nodes.
- Let $\oplus$ be a Boolean binary operator.

Return the index of the node of T corresponding to $u_1 \oplus u_2$ and expand T and H if needed.

Operations on BDDs

$\text{APPLY}[T, H](\oplus, u_1, u_2)$
return $app(u_1, u_2)$

function $app(u_1, u_2)$
if $u_1 \in \{0, 1\}$ **and** $u_1 \in \{0, 1\}$ **then**
$u \leftarrow u_1 \oplus u_2$
else if $var(u_1) = var(u_2)$ **then**
$u \leftarrow \text{MK}(var(u_1), app(low(u_1), low(u_2)), app(high(u_1), high(u_2))$
else if $var(u_1) < var(u_2)$ **then**
$u \leftarrow \text{MK}(var(u_1), app(low(u_1), u_2), app(high(u_1), u_2))$
else
$u \leftarrow \text{MK}(var(u_2), app(u_1, low(u_2)), app(u_1, high(u_2)))$
return u

Operations on BDDs

$\textsc{Apply}[T, H](\oplus, u_1, u_2)$
  $init(G)$
  return $app(u_1, u_2)$

  function $app(u_1, u_2)$
    if $G(u_1, u_2) \neq empty$ **then return** $G(u_1, u_2)$
    if $u_1 \in \{0, 1\}$ **and** $u_1 \in \{0, 1\}$ **then**
      $u \leftarrow u_1 \oplus u_2$
    else if $var(u_1) = var(u_2)$ **then**
      $u \leftarrow \textsc{Mk}(var(u_1), app(low(u_1), low(u_2)), app(high(u_1), high(u_2))$
    else if $var(u_1) < var(u_2)$ **then**
      $u \leftarrow \textsc{Mk}(var(u_1), app(low(u_1), u_2), app(high(u_1), u_2))$
    else
      $u \leftarrow \textsc{Mk}(var(u_2), app(u_1, low(u_2)), app(u_1, high(u_2)))$
    $G(u_1, u_2) \leftarrow u$
    return u