Intelligent Systems Programming

Lecture 6: **BDD Construction and Manipulation**

1. BDD construction
2. Boolean operations on BDDs
3. BDD-Based configuration

IT University of Copenhagen

Today's Program

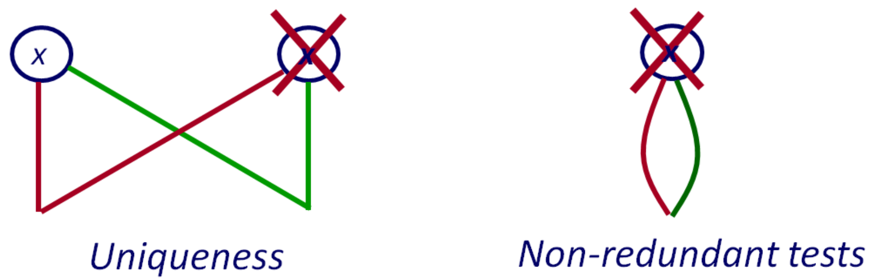
- [12:00-13:10]
  - Unique table
  - Build(t)
  - Apply(op,u_1,u_2)
- [13:20-14:00]
  - Apply example
  - BDD-Based configuration

BDD Construction

BDD construction

Last week:

1. Make a Decision Tree of the Boolean expression
2. Keep reducing it until no further reductions are possible



This week:

- Reduce the decision tree to a BDD while building it

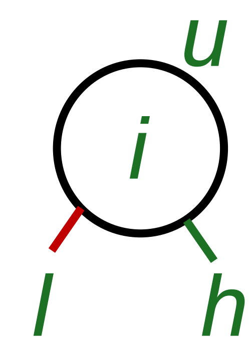
Reduce decision tree to BDD during construction

- Represent BDD by a table of unique nodes (*UT*)
- Build BDDs recursively,
  i.e. to add a new node *u:*
  1. Compute *high*(u) and *low*(u) and store them in *UT*
  2. Maintain BDD reductions when adding *u* to *UT*:
     a) Only extend *UT* with *u* if
        $high(u) \neq low(u)$ (non-redundancy test)
     b) Only extend *UT* with *u* if $u \notin UT$
        (uniqueness)

Unique Table Representation

Node Attributes

u	unique node identifier $\{\mathbf{0},\mathbf{1},2,3,\ldots\}$
i	variable index $\{1,2,\ldots,n,\boldsymbol{n+1}\}$
l	node identifier of low
h	node identifier of high



Represent Unique Table by two tables *T* and *H*

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

$H: (i,l,h) \rightarrow u$

H is the inverse of T:

$T(u) = (i,l,h) \Leftrightarrow H(i,l,h) = u$

Primitive Operations on T and H

$T : u \mapsto (i, l, h)$

$init(T)$	initialize T to contain only 0 and 1
$u \leftarrow add(T, i, l, h)$	allocate a new node u with attributes (i, l, h)
$var(u), low(u), high(u)$	lookup the attributes of u in T

$H : (i, l, h) \mapsto u$

$init(H)$	initialize H to be empty
$b \leftarrow member(H, i, l, h)$	check if (i, l, h) is in H
$u \leftarrow lookup(H, i, l, h)$	find $H(i, l, h)$
$insert(H, i, l, h, u)$	make (i, l, h) map to u in H

Unique Table Interface: MakeNode (Mk)

$\text{M}\textsc{k}[T, H](i, l, h)$

$\quad \textbf{if } l = h \textbf{ then return } l$

$\quad \textbf{else if } \mathit{member}(H, i, l, h) \textbf{ then}$

$\qquad \textbf{return } \mathit{lookup}(H, i, l, h)$

$\quad \textbf{else } u \leftarrow \mathit{add}(T, i, l, h)$

$\qquad \mathit{insert}(H, i, l, h, u)$

$\qquad \textbf{return } u$

Let’s do example on T, H and Mk!

Build

Idea: Construct the BDD recursively using the Shannon Expansion $\boldsymbol{t = x \rightarrow t[1/x], t[0/x]}$

- Terminal cases

  Build(0) = 0

  Build(1) = 1

- Recursive case

  $\text{Build}(t(x_i,x_{i+1},\ldots,x_n)) = \text{Mk}($ x_i $)$

  $\text{Build}(t(0,x_{i+1},\ldots,x_n))$ $\quad$ $\text{Build}(t(1,x_{i+1},\ldots,x_n))$

Build

$\textsc{Build}[T, H](t)$

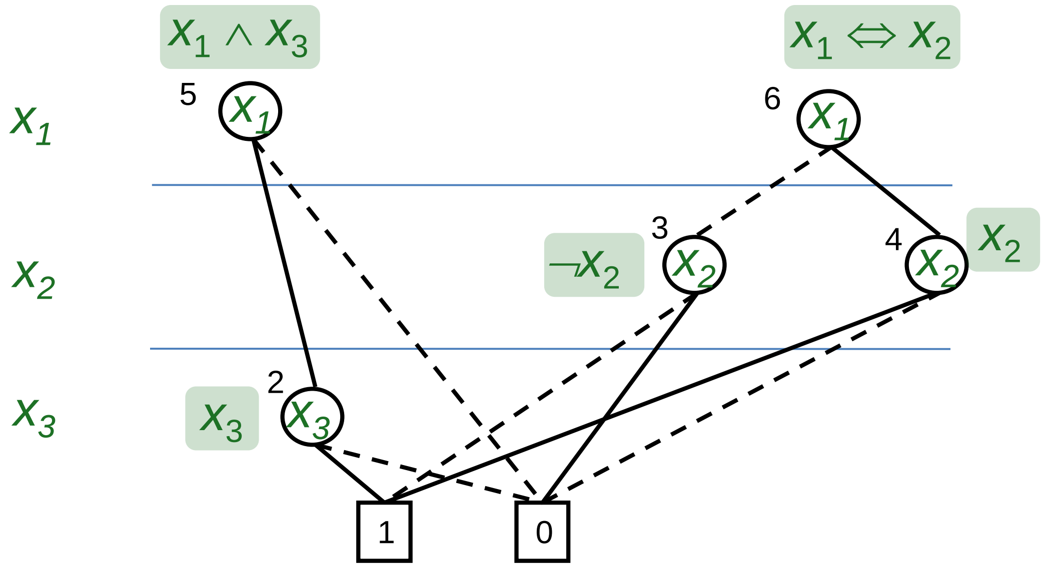
function $\textsc{build'}(t, i) =$
  if $i > n$ **then**
    if t is false **then return** 0 **else return** 1
  else $v_0 \leftarrow \textsc{build'}(t[0/x_i], i+1)$
    $v_1 \leftarrow \textsc{build'}(t[1/x_i], i+1)$
    return $\textsc{mk}(i, v_0, v_1)$
end $\textsc{build'}$

return $\textsc{build'}(t, 1)$

BDD Manipulation

Multi-Rooted BDD

Unique Table contains many BDDs



Apply

- Apply(op,u_1,u_2): computes the BDD of

$$u_1 \; op \; u_2$$

where

op : any of the 16 Boolean operators

u_1, u_2: root nodes of BDDs

- Relies on the Shannon expansion properties:

$$(x \rightarrow t_1,t_0) \; op \; (x \rightarrow t'_1,t'_0) \;\equiv\; x \rightarrow (t_1 \; op \; t'_1),(t_0 \; op \; t'_0)$$

$$(x \rightarrow t_1,t_0) \; op \; t \;\equiv\; x \rightarrow (t_1 \; op \; t),(t_0 \; op \; t)$$

Apply with $op = \wedge$

- **Terminal case:** $u \in \{0,1\}$
  $u' \in \{0,1\}$

  App($u \wedge u'$) $= u \wedge u'$

- **Recursive case:** $u = x \to u_1, u_0$
  $u' = x' \to u'_1, u'_0$

  App($u \wedge u'$) =

  Mk(x, App($u_0 \wedge u'_0$), App($u_1 \wedge u'_1$)) if $x = x'$

  Mk(x, App($u_0 \wedge u'$), App($u_1 \wedge u'$)) if $x < x'$

  Mk(x', App($u \wedge u'_0$), App($u \wedge u'_1$)) if $x' < x$

$\text{APPLY}[T, H](op, u_1, u_2)$

$init(G)$

function $\text{APP}(u_1, u_2) =$

  if $G(u_1, u_2) \neq empty$ **then return** $G(u_1, u_2)$

  else if $u_1 \in \{0, 1\}$ **and** $u_2 \in \{0, 1\}$ **then** $u \leftarrow op(u_1, u_2)$

  else if $var(u_1) = var(u_2)$ **then**

    $u \leftarrow \text{MK}(var(u_1), \text{APP}(low(u_1), low(u_2)), \text{APP}(high(u_1), high(u_2)))$

  else if $var(u_1) < var(u_2)$ **then**

    $u \leftarrow \text{MK}(var(u_1), \text{APP}(low(u_1), u_2), \text{APP}(high(u_1), u_2))$

  else $(*\ var(u_1) > var(u_2)\ *)$

    $u \leftarrow \text{MK}(var(u_2), \text{APP}(u_1, low(u_2)), \text{APP}(u_1, high(u_2)))$

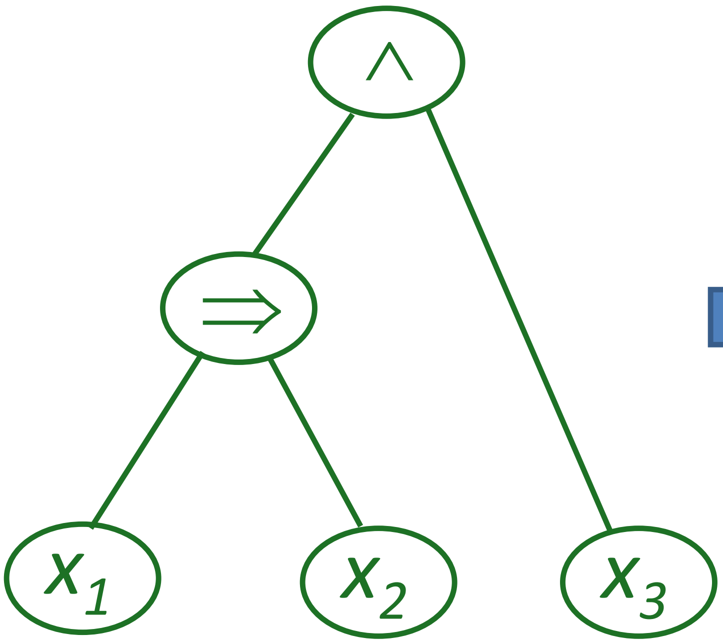
  $G(u_1, u_2) \leftarrow u$

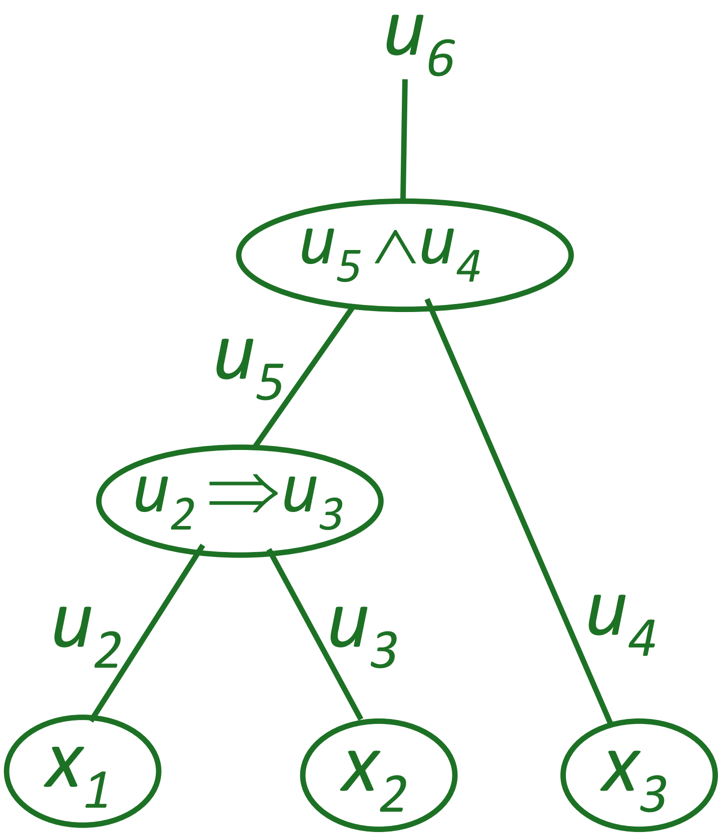
  return u

end APP

return $\text{APP}(u_1, u_2)$

Construct BDDs from expression tree





Properties of Apply

- Improvements?
  - Early termination. E.g., no reason to keep recursing if the left side in a conjunction is 0

- Complexity : $O(|u_1||u_2|)$, due to dynamic programming

- So a BDD of any formula can be computed in poly time?

BDDs

- Compact 🙂
- Equality check easy 🙂
- Easy to evaluate the truth-value of an assignment 🙂
- Boolean operations efficient 🙂
- SAT check efficient 🙂
- Tautology check efficient 🙂
- Easy to implement 🙂

BDD-Based Configuration

T-Shirt Example[1]

- $y_1 \in \{black, white, red, blue\}$: Color
  $y_2 \in \{small, medium, large\}$: Size
  $y_3 \in \{$"Men in black" - MIB,
  "Save the whales" -$STW\}$: Print

- $f_1 \equiv (y_3 = MIB) \Rightarrow (y_1 = black)$
  $f_2 \equiv (y_3 = STW) \Rightarrow (y_2 \neq small)$

[1]: Due to Erik van der Meer, now at Microsoft

Configuration Problems

A configuration problem C is a triple (Y,D,F)

- Y is a set of variables $y_1, y_2, \ldots, y_n$
- D is the Cartesian product of their finite domains $D = D_1 \times D_2 \times \ldots \times D_n$
- $F = \{f_1, f_2, \ldots, f_m\}$ is a set of propositional formulas over atomic propositions $y_i = v$, where $v \in D_i$, specifying the conditions that the variable assignments must satisfy. Each formula is inductively defined by

$$f \equiv y_i = v \mid f \wedge g \mid f \vee g \mid \neg f$$

T-Shirt Example[1]

- $y_1 \in \{black, white, red, blue\}$: Color
  $y_2 \in \{small, medium, large\}$: Size
  $y_3 \in \{$"Men in black" - MIB,
  "Save the whales" -$STW\}$: Print
- $f_1 \equiv (y_3 = MIB) \Rightarrow (y_1 = black)$
  $f_2 \equiv (y_3 = STW) \Rightarrow (y_2 \neq small)$
-

y_1	y_2	y_3
black	small	MIB
white	medium	STW
red	large	
blue		

[1]: Due to Erik van der Meer, now at Microsoft

T-Shirt Example[1]

- $y_1 \in \{black, white, red, blue\}$: Color
  $y_2 \in \{small, medium, large\}$: Size
  $y_3 \in \{$"Men in black" - MIB, "Save the whales" - $STW\}$: Print

- $f_1 \equiv (y_3 = MIB) \Rightarrow (y_1 = black)$
  $f_2 \equiv (y_3 = STW) \Rightarrow (y_2 \neq small)$

y_1	y_2	y_3
black	small	MIB
white	medium	STW
red	large	
blue		

[1]: Due to Erik van der Meer, now at Microsoft

T-Shirt Example[1]

- $y_1 \in \{black, white, red, blue\}$: Color
  $y_2 \in \{small, medium, large\}$: Size
  $y_3 \in \{$"Men in black" - MIB,
  "Save the whales" -$STW\}$: Print

- $f_1 \equiv (y_3 = MIB) \Rightarrow (y_1 = black)$
  $f_2 \equiv (y_3 = STW) \Rightarrow (y_2 \neq small)$

y_1	y_2	y_3
black	small	MIB
white	medium	STW
red	large	
blue		

[1]: Due to Erik van der Meer, now at Microsoft

T-Shirt Example[1]

- $y_1 \in \{black, white, red, blue\}$: Color
  $y_2 \in \{small, medium, large\}$: Size
  $y_3 \in \{$"Men in black" - MIB,
  "Save the whales" -$STW\}$: Print
- $f_1 \equiv (y_3 = MIB) \Rightarrow (y_1 = black)$
  $f_2 \equiv (y_3 = STW) \Rightarrow (y_2 \neq small)$
-

y_1	y_2	y_3
black	small	MIB
white	medium	STW
red	large	
blue		

[1]: Due to Erik van der Meer, now at Microsoft

Interactive Product Configurator

IPC(C)

1. $R \leftarrow$ COMPILE(C)
2. **while** $|R| > 1$ **do**
3. **choose** $(y_i = v) \in$ VALIDASSIGNMENTS(R)
4. $R \leftarrow R \wedge (y_i = v)$

BDD-based configuration

Idea

1. Use a BDD to represent *R*
2. Use a polynomial-time BDD algorithm to compute VALIDASSIGNMENTS(*R*)

Represent *R* by a BDD

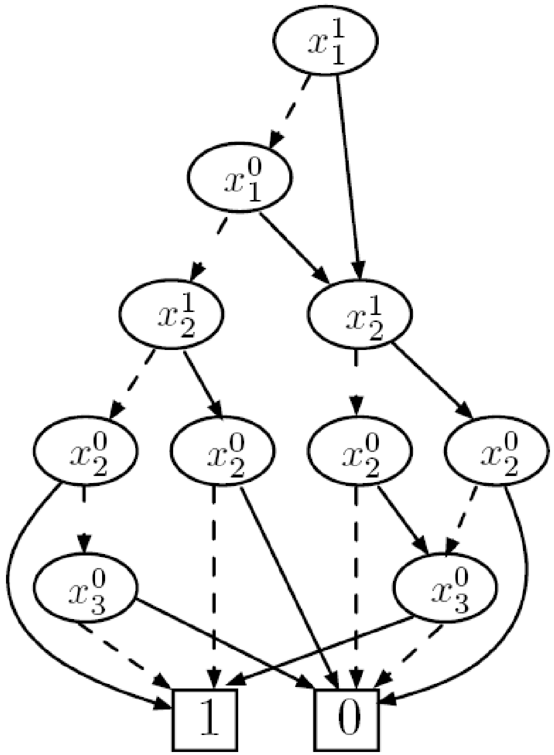
1. Define domains in binary

   (x_1^1, x_1^0): (0,0) = *black*, (0,1) = *white*, (1,0) = *red*, (1,1) = *blue*

   (x_2^1, x_2^0): (0,0) = *small*, (0,1) = *medium*, (1,1) = *large*

   x_3^0 : 0 = *MIB*, 1 = *STW*

2. Build a BDD of the rules



Compute VALIDASSIGNMENTS(R)

- Trace paths for each variable layer in the BDD