Implementation of ROBDD

data structure

- Hashtable
- List
- bool expression

operations

- MK(i,l,h)
- Build(t)
- Apply(op, u1,u2)
- Restrict(u, j, b)
- SatCount(u)
- AnySat(u)
- AllSat(u)

Mk

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

```
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, u)
        return u
```

Figure 8: The function $\textsc{mk}[T, H](i, l, h)$.

Mk cont.

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

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

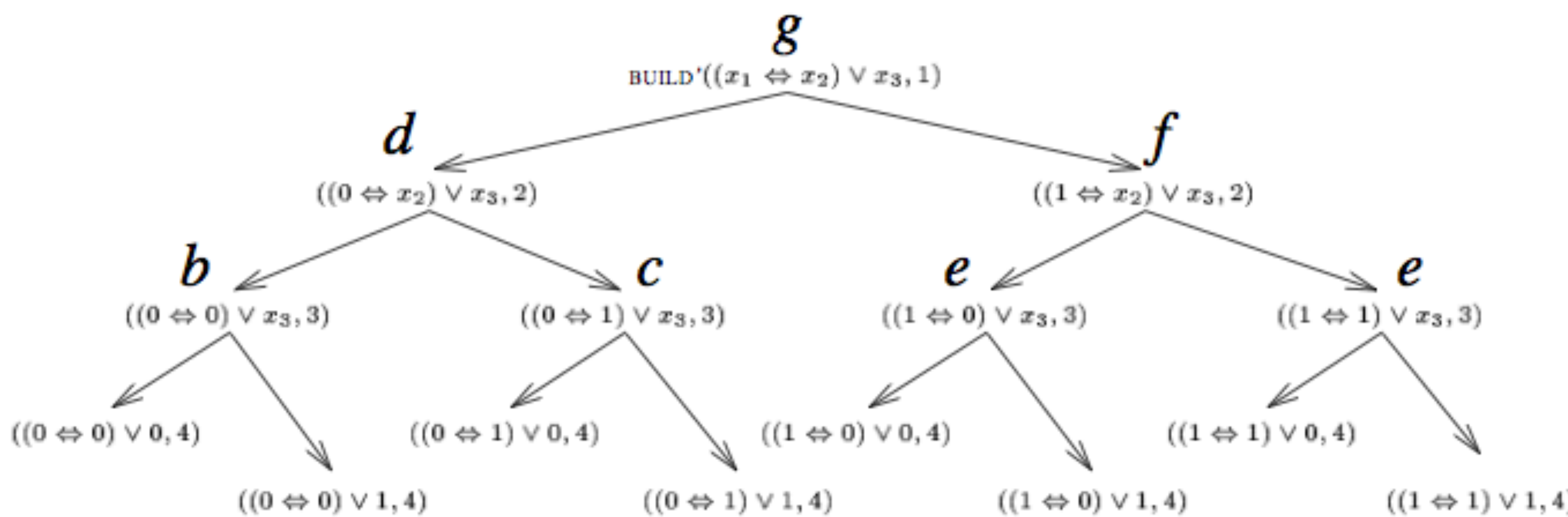
Build

BUILD$[T, H](t)$

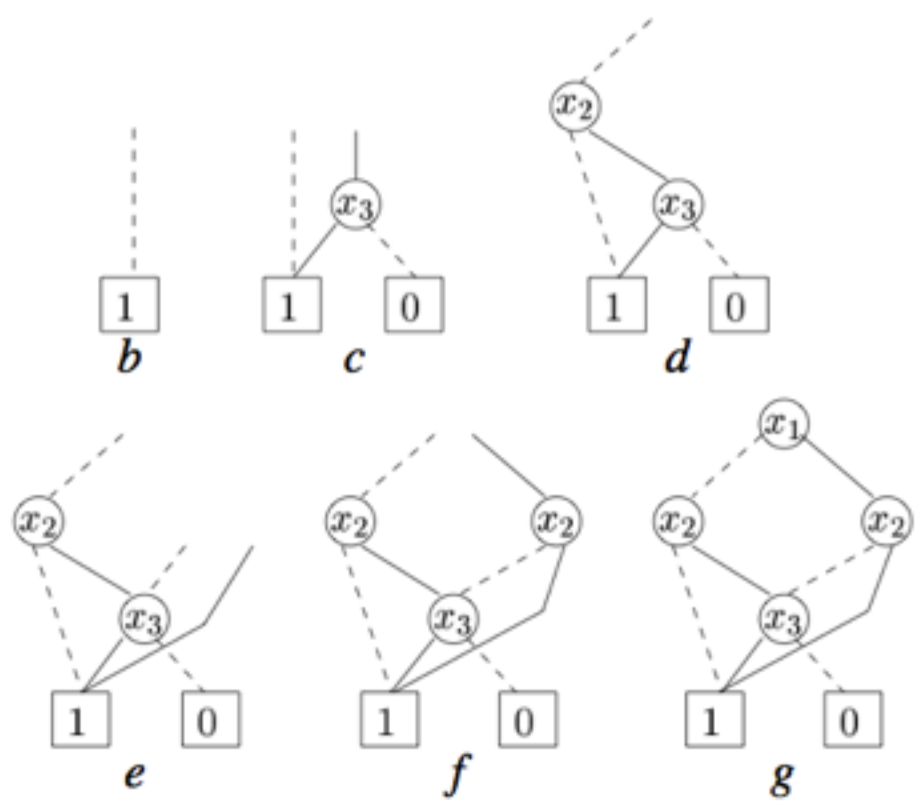
function BUILD'(t, i) =
    if $i > n$ **then**
        if t is false **then return** 0 **else return** 1
    else $v_0 \leftarrow$ BUILD'$(t[0/x_i], i+1)$
        $v_1 \leftarrow$ BUILD'$(t[1/x_i], i+1)$
        return MK(i, v_0, v_1)
end BUILD'

return BUILD'$(t, 1)$

Figure 9: Algorithm for building an ROBDD from a Boolean expression t using the ordering $x_1 < x_2 < \cdots < x_n$. In a call BUILD'(t, i), i is the lowest index that any variable of t can have. Thus when the test $i > n$ succeeds, t contains no variables and must be either constantly false or true.


g
BUILD'$((x_1 \Leftrightarrow x_2) \vee x_3, 1)$
d
$((0 \Leftrightarrow x_2) \vee x_3, 2)$
f
$((1 \Leftrightarrow x_2) \vee x_3, 2)$
b
$((0 \Leftrightarrow 0) \vee x_3, 3)$
c
$((0 \Leftrightarrow 1) \vee x_3, 3)$
e
$((1 \Leftrightarrow 0) \vee x_3, 3)$
e
$((1 \Leftrightarrow 1) \vee x_3, 3)$
$((0 \Leftrightarrow 0) \vee 0, 4)$
$((0 \Leftrightarrow 0) \vee 1, 4)$
$((0 \Leftrightarrow 1) \vee 0, 4)$
$((0 \Leftrightarrow 1) \vee 1, 4)$
$((1 \Leftrightarrow 0) \vee 0, 4)$
$((1 \Leftrightarrow 0) \vee 1, 4)$
$((1 \Leftrightarrow 1) \vee 0, 4)$
$((1 \Leftrightarrow 1) \vee 1, 4)$

a


x_1
x_2
x_3
1
0
b
c
d
e
f
g

Apply

```
APPLY[T, H](op, u1, u2)
init(G)

function APP(u1, u2) =
  if G(u1, u2) ≠ empty then return G(u1, u2)
  else if u1 ∈ {0, 1} and u2 ∈ {0, 1} then u ← op(u1, u2)
  else if var(u1) = var(u2) then
        u ← MK(var(u1), APP(low(u1), low(u2)), APP(high(u1), high(u2)))
  else if var(u1) < var(u2) then
        u ← MK(var(u1), APP(low(u1), u2), APP(high(u1), u2))
  else (* var(u1) > var(u2) *)
        u ← MK(var(u2), APP(u1, low(u2)), APP(u1, high(u2)))
  G(u1, u2) ← u
  return u
end APP

return APP(u1, u2)
```

Restrict

RESTRICT$[T, H](u, j, b)$ =
function *res*(u) =
    if $var(u) > j$ **then return** u
    else if $var(u) < j$ **then return** MK$(var(u),$ *res*$(low(u)),$ *res*$(high(u)))$
    else (* $var(u) = j$ *) **if** $b = 0$ **then return** *res*$(low(u))$
    else (* $var(u) = j, b = 1$ *) **return** *res*$(high(u))$
end *res*
return *res*(u)

Figure 13: The algorithm RESTRICT$[T, H](u, j, b)$ which computes an ROBDD for $t^u[j/b]$.

SatCount

$\textsc{SatCount}[T](u)$

function *count*(u)
    if $u = 0$ **then** *res* $\leftarrow 0$
    else if $u = 1$ **then** *res* $\leftarrow 1$
    else *res* $\leftarrow \; 2^{var(low(u)) - var(u) - 1} * count(low(u)) + 2^{var(high(u)) - var(u) - 1} * count(high(u))$
    return *res*
end *count*

return $2^{var(u)-1} * count(u)$

Figure 14: An algorithm for determining the number of valid truth assignments. Recall, that the "variable index" var of 0 and 1 in the ROBDD representation is $n + 1$ when the ordering contains n variables (numbered 1 through n). This means that *var*(0) and *var*(1) always gives $n + 1$.

AnySat

$\text{ANYSAT}(u)$

if $u = 0$ **then** Error
else if $u = 1$ **then return** $[]$
else if $low(u) = 0$ **then return** $[x_{var(u)} \mapsto 1, \text{ANYSAT}(high(u))]$
else return $[x_{var(u)} \mapsto 0, \text{ANYSAT}(low(u))]$

Figure 15: An algorithm for returning a satisfying truth-assignment. The variables are assumed to be $x_1, \ldots, x_n$ ordered in this way.

AllSat

$\textsc{AnySat}(u)$

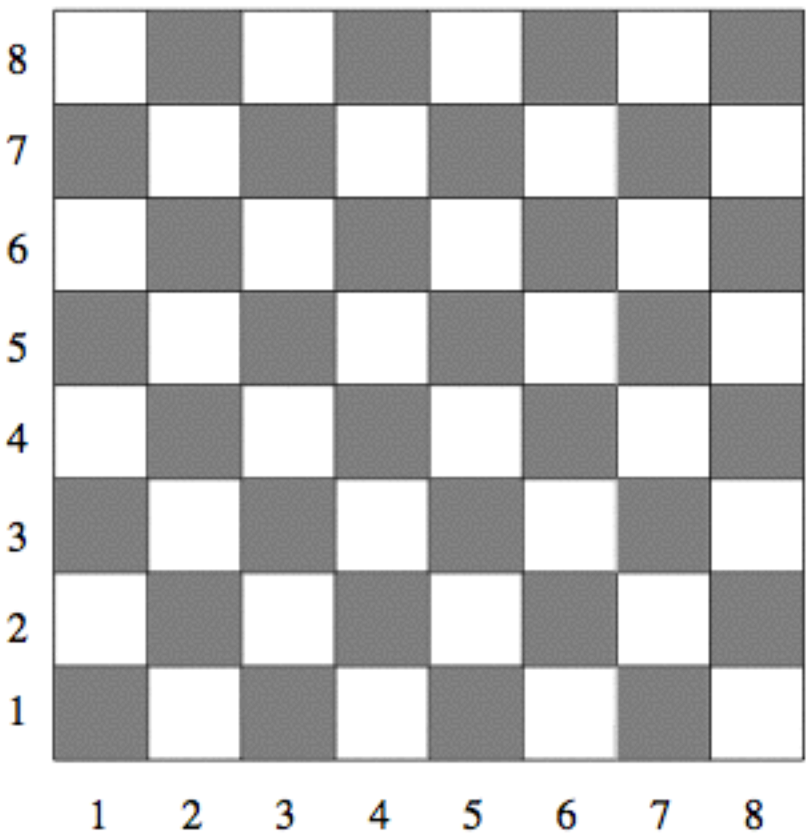
if $u = 0$ **then** Error
else if $u = 1$ **then return** $[\,]$
else if $low(u) = 0$ **then return** $[x_{var(u)} \mapsto 1, \textsc{AnySat}(high(u))]$
else return $[x_{var(u)} \mapsto 0, \textsc{AnySat}(low(u))]$

Figure 15: An algorithm for returning a satisfying truth-assignment. The variables are assumed to be $x_1, \ldots, x_n$ ordered in this way.

Test of robbd

- Test of hashtable
- Test of bool expression
- Test of Build
- Test of Apply
- Test of AllSat, AnySat, SatCount

N Queens problem



For all i

$$x_{i1} \vee x_{i2} \vee \cdots \vee x_{iN}$$

$$x_{ij} \Rightarrow \bigwedge_{1 \leq l \leq N, l \neq j} \neg x_{il}$$

$$x_{ij} \Rightarrow \bigwedge_{1 \leq k \leq N, k \neq i} \neg x_{kj}$$

$$x_{ij} \Rightarrow \bigwedge_{1 \leq k \leq N, 1 \leq j+k-i \leq N, k \neq i} \neg x_{k,j+k-i}$$

$$x_{ij} \Rightarrow \bigwedge_{1 \leq k \leq N, 1 \leq j+i-k \leq N, k \neq i} \neg x_{k,j+i-k}$$

Taking the conjunction of all the above requirements, we get a predicate $Sol_N(\vec{x})$ true at exactly the configurations that are solutions to the N queens problem.