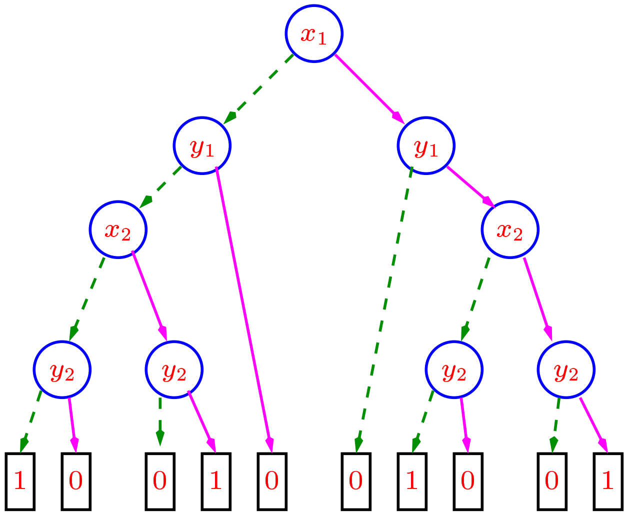
BDD's

We start with a binary decision diagram. For example, following is a decision diagram (tree) for the formula $(x_1 = y_1) \land (x_2 = y_2)$:

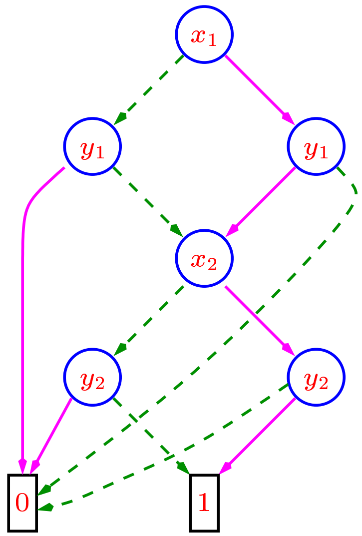


In general, it requires an exponential number of nodes.

Optimize

- Identify identical subgraphs.
- Remove redundant tests.

Yielding:



Definitions

A binary decision diagram BDD is a rooted, directed acyclic graph with

- One or two nodes of out-degree zero labeled 0 or 1, and
- A set of variable nodes u of out-degree 2. The two outgoing edges are given by the functions $low(u)$ and $high(u)$. A variable $var(u)$ is associated with each node.

A BDD is ordered (OBDD) if the variables respect a given linear order $x_1 < x_2 < \cdots < x_n$ on all paths through the graph. An OBDD is reduced (ROBDD) if it satisfies:

- Uniqueness – no two distinct nodes are the roots of isomorphic subgraphs.
- No redundant tests – $low(u) \neq high(u)$ for all nodes u in the graph.

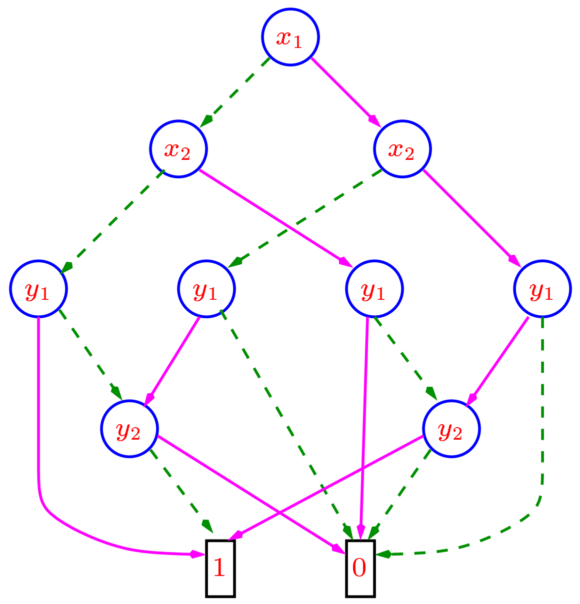
For simplicity, we will refer to ROBDD simply as BDDs.

Course G22.3033.007 Lecture 5 R. Dewar and A. Pnueli

Canonicity

Claim 4. *For every function f : **Bool**$^n \rightarrow$ **Bool** and variable ordering $x_1 < x_2 < \cdots < x_n$, there exists exactly one BDD representing this function.*

The complexity of BDD representation is very sensitive to the variable ordering. For example, the BDD representation of $(x_1 = y_1) \wedge (x_2 = y_2)$ under the variable ordering $x_1 < x_2 < y_1 < y_2$ is:



Implementation of BDD Packages

Types and Variables:

$$
\begin{array}{rcl}
node & = & \textbf{naturals} \\
var\_num & = & \textbf{naturals} \\
node\_rec & = & \left[\begin{array}{lll} \textbf{record of} & & \\ \quad var & : & var\_num; \\ \quad low, high & : & node \\ \textbf{end\_record} & & \end{array}\right] \\
T & : & node \to node\_rec \\
H & : & node\_rec \to node \cup \{\perp\}
\end{array}
$$

Operations:

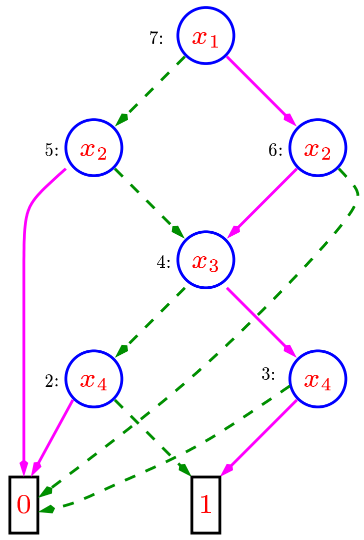
$init(T)$	Initialize T to contain only 0 and 1
$u := new(T, i, \ell, h)$	allocate a new node u, such that $T(u) = \langle i, \ell, h \rangle$
$init(H)$	initialize H to $\perp$

H is the inverse of T. That is, $H(T(u)) = u$, for every $u \in dom(T)$.

We will write $var(u)$, $low(u)$, $high(u)$, and $H(i, \ell, h)$ as abbreviations for $T(u).var$, $T(u).low$, $T(u).high$, and $H(\langle i, \ell, h \rangle)$.

Course G22.3033.007 Lecture 5 R. Dewar and A. Pnueli

Internal Representation



$T : u \to \langle i, \ell, h \rangle$

u	var	low	$high$
0			
1			
2	4	1	0
3	4	0	1
4	3	2	3
5	2	4	0
6	2	0	4
7	1	5	6

Making or Retrieving a node_id

Function $\textsc{Mk}\ (i : \mathit{var\_num}; \ell, h : \mathit{node}) : \mathit{node}$

– – Make or retrieve a node with attributes (i, ℓ, h)

if $\ell = h$ **then return** ℓ

if $H(i, \ell, h) \neq \bot$ **then return** $H(i, \ell, h)$

$u := \mathit{new}(i, \ell, h)$

$H(i, \ell, h) := u$

return u

Applying a Binary Boolean Operation to two BDD's

Let op : **Bool** $\times$ **Bool** $\rightarrow$ **Bool** be a binary boolean operation. The following function uses the auxiliary dynamic array $G : \mathit{node} \times \mathit{node} \rightarrow \mathit{node}$.

Function Apply $(\mathit{op}\ ; u_1, u_2 : \mathit{node}) : \mathit{node}$

– – Apply op to BDD's u_1 and u_2

$G := \bot$

function $\mathit{App}(u_1, u_2 : \mathit{node}) : \mathit{node} \quad =$

  if $G[u_1, u_2] \neq \bot$ **then return** $G[u_1, u_2]$

  if $u_1 \in \{0, 1\} \ \wedge \ u_2 \in \{0, 1\}$ **then** $u := \mathit{op}(u_1, u_2)$

  else if $\mathit{var}(u_1) = \mathit{var}(u_2)$ **then**

    $u := \textsc{Mk}(\mathit{var}(u_1), \mathit{App}(\mathit{low}(u_1), \mathit{low}(u_2)), \mathit{App}(\mathit{high}(u_1), \mathit{high}(u_2)))$

  else if $\mathit{var}(u_1) < \mathit{var}(u_2)$ **then**

    $u := \textsc{Mk}(\mathit{var}(u_1), \mathit{App}(\mathit{low}(u_1), u_2), \mathit{App}(\mathit{high}(u_1), u_2))$

  else $(* \mathit{var}(u_1) > \mathit{var}(u_2) *)$

    $u := \textsc{Mk}(\mathit{var}(u_2), \mathit{App}(u_1, \mathit{low}(u_2)), \mathit{App}(u_1, \mathit{high}(u_2)))$

  $G[u_1, u_2] := u$

  return u

end App

return $\mathit{App}(u_1, u_2)$

Restriction (Substitution)

Function $\text{REST}\,(u : node;\; j : var\_num;\; b :$ **Bool**$) : node$
−− Substitute b for x_j in BDD u

$G := \bot$

function $res(u : node) : node \quad =$
  if $G[u] \neq \bot$ **then return** $G[u]$
  if $var(u) > j$ **then** $r := u$
  else if $var(u) < j$ **then**
    $r := \text{MK}(var(u), res(low(u)), res(high(u)))$
  else $(* var(u) = j *)$ **if** $b = 0$ **then** $r := low(u)$
                else $r := high(u)$
  $G[u] := r$
  return r
end res

return $res(u)$

Restriction is the same as substitution. We denote by $t[x \mapsto b]$ the result of substituting b for x in assertion t.

Quantification

Existential quantification can be computed, using the equivalence

$$\exists x : t \quad \sim \quad t[x \mapsto 0] \;\vee\; t[x \mapsto 1]$$

Universal quantification can be computed dually:

$$\forall x : t \quad \sim \quad t[x \mapsto 0] \;\wedge\; t[x \mapsto 1]$$

Application to Symbolic Model Checking

Let V be the state variables for the FDS $\mathcal{D}$. Taking a vocabulary $U = V \cup V'$, we represent the state formulas Θ, J for each $J \in \mathcal{J}$, p_i, q_i, for each $\langle p_i, q_i \rangle \in \mathcal{C}$, and the SMC-INV symbolic working variables new and old as BDD's over U which are independent of V'.

The transition relation ρ is represented as a BDD over U which may be fully dependent on both V and V'.

All the boolean operations used in the SMC-INV algorithm can be implemented by the Apply function. Negation can be computed by $\neg t = t \oplus 1$, where $\oplus$ is sum modulo 2.

To check for equivalence such as $old = new$ we compute $t := (old \leftrightarrow new)$ and then verify that the result is the singleton BDD 1.

The existential pre-condition transformer is computed by

$$\rho \diamond \psi \quad = \quad \exists V' : \rho(V, V') \wedge \psi(V')$$

Priming an assertion ψ is performed by

$$prime(\psi) = \exists V : \psi(V) \ \wedge \ V' = V$$