Node Minimization

Problem:
Given a Boolean network, optimize it by minimizing each node as much as possible.

Note:

- The initial network structure is given.
  Typically applied after the global optimization, i.e. division and resubstitution.

- We minimize the function associated with each node.
  What do we mean by minimizing the node "*as much as possible*"?

Functions Implementable at a Node

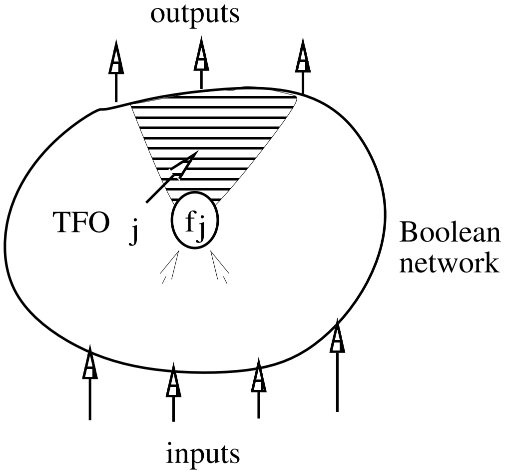
In a Boolean network, we may represent a node using the primary inputs $\{x_1, \ldots, x_n\}$ + the intermediate variables $\{y_1, \ldots, y_m\}$, as long as the network is acyclic.

DEFINITION 1 *A function f_j, whose variables are a subset of $\{x_1, \ldots, x_n, y_1, \ldots, y_m\}$, is* **implementable** *at a node j if*

1. *the variables of f_j does not intersect with TFO_j and*

2. *the replacement of the function associated with j by f_j does not change the functionality of the network.*

The set of implementable functions at j provides the solution space of the local optimization at node j.



$TFO_j = \{$ node i s.t. $i = j$ or $\exists$ path from j to $i\}$.

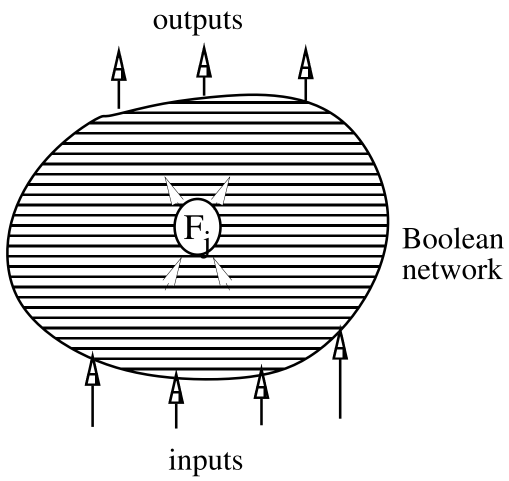
Prime and Irredundant Boolean Network

Consider a sum-of-products expression F_j associated with a node j.

DEFINITION 2 *F_j is* **prime** *(in multi-level sense) if for all cubes $c \in F_j$, no literal of c can be removed without changing the functionality of the network.*

DEFINITION 3 *F_j is* **irredundant** *if for all cubes $c \in F_j$, the removal of c from F_j changes the functionality of the network.*

DEFINITION 4 *A Boolean network is* **prime and irredundant** *if F_j is prime and irredundant for all j.*



THEOREM 1 *A network is 100% testable for single stuck-at faults (s-a-0 or s-a-1) iff it is prime and irredundant.*

Local Optimization

Goals:
Given a Boolean network,

1. make the network prime and irredundant.

2. for a given node of the network, find a least-cost sum-of-products expression among the implementable functions at the node.

Note:
Goal 2 implies the goal 1, i.e.
our attempt is more than just 100% testability. There are many expressions that are prime and irredundant. We seek the best.

Key Ingredient - Don't Cares:

- External don't cares - XDC_k, $k = 1, \ldots, p$.
  - minterms of the primary inputs given for each primary output

- Internal don't cares - derived from the structure
  1. Satisfiability - SDC
  2. Observability - ODC

SDC

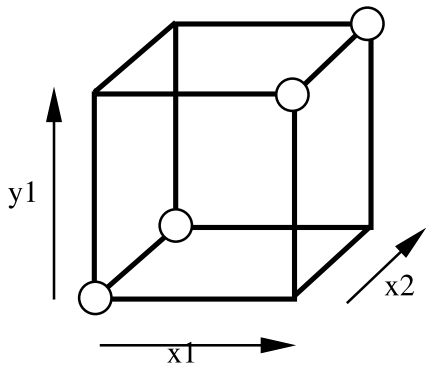
Recall:
We may represent a node using the primary inputs + the intermediate variables.
- The Boolean space is B^{n+m}.

However, the intermediate variables are dependent on the primary inputs.

Thus not all the minterms of B^{n+m} *can occur*.
Example:

$$\begin{aligned} y_1 &= F_1 = \overline{x_1} \\ y_j &= F_j = y_1 x_2 \end{aligned}$$

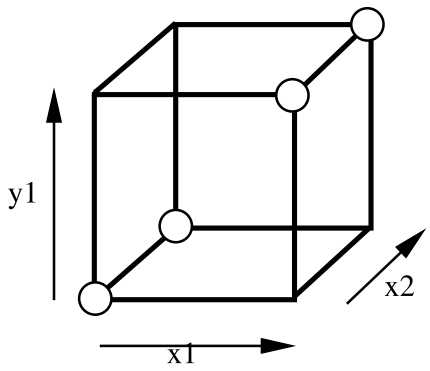
Since $y_1 = \overline{x_1}$, $y_1 \oplus \overline{x_1}$ never occurs.



Thus we may include these points to represent F_j.
$\Rightarrow$ Don't Cares

SDC

$$
\begin{aligned}
y_1 &= F_1 = \overline{x_1} \\
y_j &= F_j = y_1 x_2
\end{aligned}
$$



$SDC = (y_1 \oplus \overline{x_1}) + (y_j \oplus y_1 x_2)$

In general,

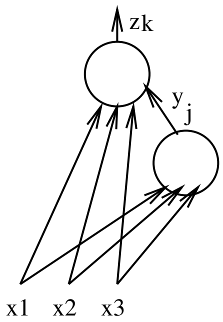
$$SDC = \sum_{j=1}^{m} (y_j \overline{F_j} + \overline{y_j} F_j)$$

Note : $SDC \subseteq B^{n+m}$

ODC

Example:

$$\begin{aligned} y_j &= \overline{x_1}x_2 + x_1\overline{x_3} \\ z_k &= x_1x_2 + y_j\overline{x_2} + \overline{y_j x_3} \end{aligned}$$



Any minterm of $x_1x_2 + \overline{x_2x_3} + x_2x_3$ determines z_k independent of y_j.

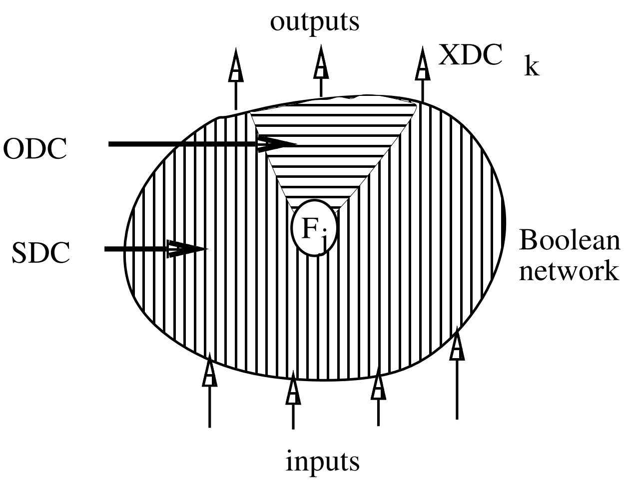
The ODC of y_j for z_k is the set of minterms of the primary inputs for which the value of y_j is not observable at z_k.
- $ODC_{jk} = \{\mathbf{x} \in B^n$ s.t. $z_k(\mathbf{x}) \mid_{y_j=0} = z_k(\mathbf{x}) \mid_{y_j=1}\}$

This means that the two Boolean networks, one with y_j forced to 0 and one with y_j forced to 1 compute the same for $x \in ODC_{jk}$.

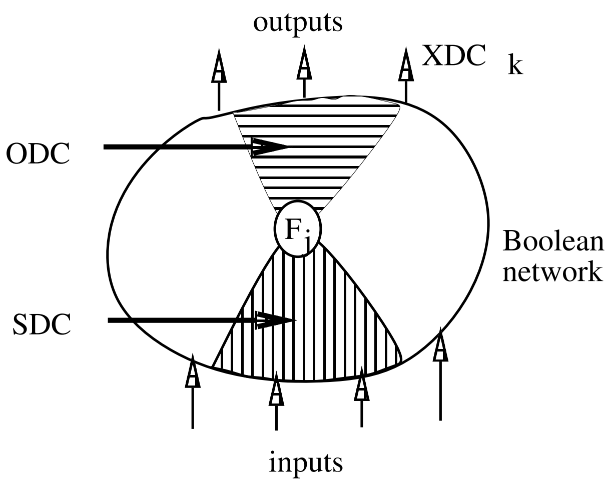
Don't Cares for Node j

Define the don't care sets DC_j and DI_j for a node j as

$$
\begin{aligned}
DC_j &= \sum_{i \notin TFO_j} (y_i \overline{F_i} + \overline{y_i} F_i) + \prod_{k=1}^{p} (ODC_{jk} + XDC_k), \\
DI_j &= \sum_{i \in TFI_j - PI} (y_i \overline{F_i} + \overline{y_i} F_i) + \prod_{k=1}^{p} (ODC_{jk} + XDC_k).
\end{aligned}
$$



DC j



DI j

$TFI_j = \{\text{node } i \text{ s.t. } \exists \text{ path from } i \text{ to } j\}$

Main Theorem - on DC_j

THEOREM 2 *The function* $f_C = (F_j - DC_j, DC_j, \overline{F_j + DC_j})$ *is the complete set of implementable functions at node* j.

COROLLARY 3 F_j *is prime and irredundant (in the multi-level sense) iff it is prime and irredundant cover of* f_C.

- A least-cost expression at node j can be obtained by minimizing f_C.
- A prime and irredundant Boolean network can be obtained by using only 2-level logic minimization for each node j with the don't care DC_j.

Note:
If F_j is changed, then DC_i may change for some other node i in the network.

Main Theorem - on DI_j

THEOREM 4 *F_j is prime and irredundant (in multi-level sense) iff it is a prime and irredundant cover of $f_I = (F_j - DI_j, DI_j, \overline{F_j + DI_j})$.*

Note that this does not contradict the previous theorem, since if F_j has a variable not in TFI, then F_j is not a cover of f_I. Note also that here DI_j is referring to TFI of F_j.

Proof: Let η be the given Boolean network. The primality or the irredundancy of F_j can be checked by eliminating literals or cubes, which leads to a new network $\tilde{\eta}$ where, because only removal was done, $TFI_j(\tilde{\eta}) \subseteq TFI_j(\eta)$. The following lemma claims the theorem.

LEMMA 1 *Given a Boolean network η and a node j, let $\tilde{\eta}$ be the network obtained by replacing F_j with $\tilde{F}_j$, where $TFI_j(\tilde{\eta}) \subseteq TFI_j(\eta)$. Then*

$$\eta \equiv_{XDC} \tilde{\eta} \Leftrightarrow \forall(\mathbf{x}, \mathbf{y}) \in B^{n+m}, F_j(\mathbf{x}, \mathbf{y}) = \tilde{F}_j(\mathbf{x}, \mathbf{y}) \text{ or } (\mathbf{x}, \mathbf{y}) \in DI_j.$$

■

Thus instead of DC_j, we can use DI_j to obtain a prime and irredundant Boolean network.

DC_j v.s. DI_j

$DC_j \supseteq DI_j$, and in general,
there may be an implementation obtained by DC_j but not by DI_j.

Example:

$$\begin{aligned} z_1 &= y_i &&= x_1\overline{x_2}x_3 = F_i \\ z_2 &= y_j &&= \overline{x_1}x_4 + x_2x_4 + \overline{x_3} = F_j \end{aligned}$$

$$\begin{aligned} DC_j &= y_i \oplus x_1\overline{x_2}x_3 \\ DI_j &= \phi \end{aligned}$$

With DC_j, we obtain $y_j = \overline{y_i}x_4 + \overline{x_3} = \widetilde{F}_j$.

Note: This corresponds to Boolean resubstitution.

[Paradox? - No]

- In general, there is an implementation obtained by f_C but not by f_I.
- Corollary 3 and Theorem 4 imply that F_j is prime and irredundant in f_C iff it is prime and irredundant in f_I.

Summary

- Two don't cares are defined; $DI_j \subseteq DC_j$.

- The set of implementable functions at j is represented by the incompletely specified function f_C.
  $\Rightarrow$ a least-cost implementation can be found with DC_j

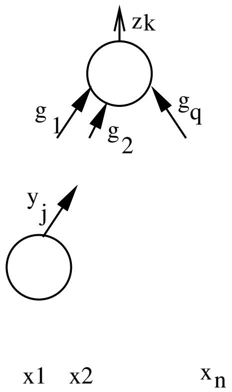
- A prime and irredundant expression in multi-level sense can be obtained even with f_I.
  $\Rightarrow$ a prime and irredundant network is obtained even with DI_j.

- Our goals are achieved by using only 2-level logic minimization techniques with appropriate don't cares for each node of the network.

Local Optimization - Practical Questions

1. How to compute the don't care set at a node?
   - XDC is given, SDC is easy,
     but how about ODC?
   - The don't care may be too large.
     A good subset?
2. How to minimize the funtion with the don't care?
   - Complement may be too large.
   - #Literals v.s. #Cubes.

ODC Computation



$ODC_{jk} = \{\mathbf{x} \in B^n \text{ s.t. } z_k(\mathbf{x}) \mid_{y_j=0} = z_k(\mathbf{x}) \mid_{y_j=1}\}$

Denote $ODC_{jk} = \overline{(\frac{\partial z_k}{\partial y_j})}$,
where $\frac{\partial z_k}{\partial y_j} = (z_k(\mathbf{x}) \mid_{y_j=0}) \oplus (z_k(\mathbf{x}) \mid_{y_j=1})$.

In general, when z_k is not explicitly dependent on y_j, $\frac{\partial z_k}{\partial y_j}$ is complicated to compute:

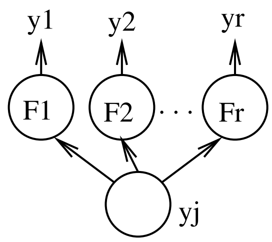
$$\begin{aligned}
\frac{\partial z_k}{\partial y_j} \quad = \quad & \frac{\partial z_k}{\partial g_1} \cdot \frac{\partial g_1}{\partial y_j} \oplus \frac{\partial z_k}{\partial g_2} \cdot \frac{\partial g_2}{\partial y_j} \oplus \ldots \oplus \frac{\partial z_k}{\partial g_q} \cdot \frac{\partial g_q}{\partial y_j} \\
& \oplus \frac{\partial^2 z_k}{\partial g_1 g_2} \cdot \frac{\partial g_1}{\partial y_j} \cdot \frac{\partial g_2}{\partial y_j} \oplus \frac{\partial^2 z_k}{\partial g_1 \partial g_3} \cdot \frac{\partial g_1}{\partial y_j} \cdot \frac{\partial g_3}{\partial y_j} \oplus \ldots \\
& \oplus \frac{\partial^2 z_k}{\partial g_1 \partial g_q} \cdot \frac{\partial g_1}{\partial y_j} \cdot \frac{\partial g_q}{\partial y_j} \oplus \ldots \oplus \frac{\partial^2 z_k}{\partial g_{q-1} \partial g_q} \cdot \frac{\partial g_{q-1}}{\partial y_j} \cdot \frac{\partial g_q}{\partial y_j} \oplus \\
& \frac{\partial^3 z_k}{\partial g_1 \partial g_2 \partial g_3} \cdot \frac{\partial g_1}{\partial y_j} \cdot \frac{\partial g_2}{\partial y_j} \cdot \frac{\partial g_3}{\partial y_j} \oplus \ldots \oplus \frac{\partial^q z_k}{\partial g_1 \partial g_2 \ldots \partial g_q} \cdot \frac{\partial g_1}{\partial y_j} \ldots \frac{\partial g_q}{\partial y_j}.
\end{aligned}$$

ODC Computation

$$ODC_j = \prod_{k=1}^{p} ODC_{jk}$$

Conjecture:

$$ODC_j = \prod_{i \in FO_j} [(F_i \mid_{y_j=0} \overline{\oplus} F_i \mid_{y_j=1}) + ODC_i]$$



This conjecture is true if there is no reconvergent fanout in TFO_j.

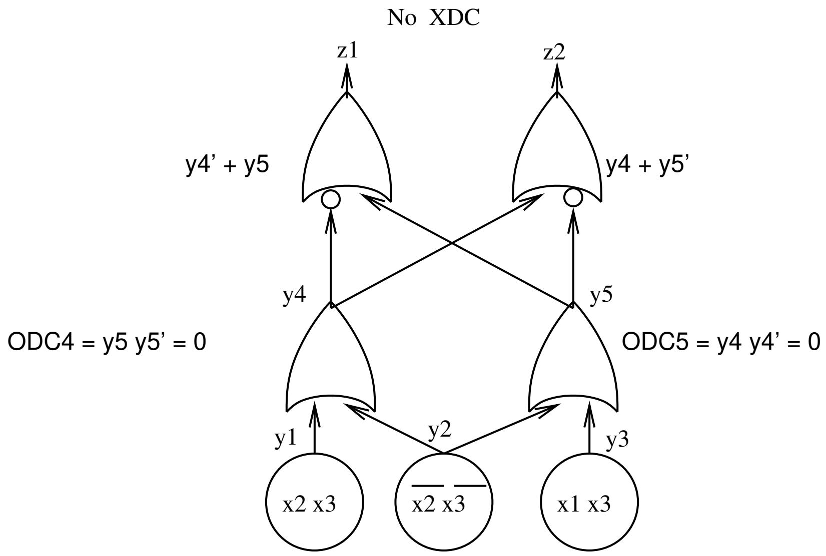
With a reconvergence, the conjecture can be incorrect in two ways:

1. it does not compute the complete ODC.
   (still have correct results)

2. it contains care points.
   (leads to incorrect answer)

Incorrectness with Reconvergent Fanouts

Example 1:

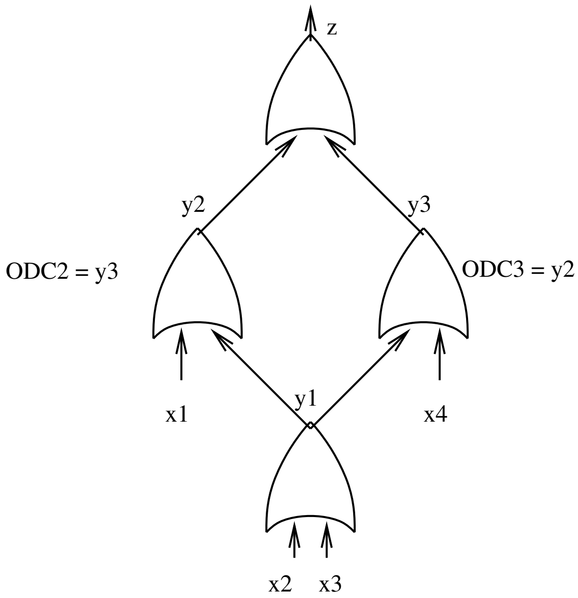


The correct $ODC_2 = x_1 x_2 + \overline{x_1 x_2} + \overline{x_3}$

$$(F_4 \mid_{y_2=0} \overline{\oplus} F_4 \mid_{y_2=1} + ODC_4)(F_5 \mid_{y_2=0} \overline{\oplus} F_5 \mid_{y_2=1} + ODC_5)$$
$$= y_1 y_3 = x_1 x_2 x_3$$

Incorrectness with Reconvergent Fanouts

Example 2:



The correct $ODC_1 = x_1 + x_4$

$$(F_2 \mid_{y_1=0} \overline{\oplus} F_2 \mid_{y_1=1} + ODC_2)(F_3 \mid_{y_1=0} \overline{\oplus} F_3 \mid_{y_1=1} + ODC_3)$$
$$= (x_1 + y_3)(x_4 + y_2) = x_1 + x_2 + x_3 + x_4$$

ODC Computation

- Transduction
  - ODC's for NOR-gate Networks
  - Interesting Subset of ODC's
- Extension of Transduction for General Networks
- Direct Computations using BDD's

Transduction

- Given a node j, a **permissible function** at j is a function of the primary inputs implementable at j.

- Transduction computes a set of permissible functions for a NOR gate in an all NOR-gate network.
  - MSPF (Maximum Set of Permissible Functions) - $f_j^M = \mathcal{F}_j^M$. (We will see that this is an incompletely specified function).
  - CSPF (Compatible Set of Permissible Functions) - $f_j^C = \mathcal{F}_j^C$

We denote by g_j the function F_j expressed in terms of the primary inputs.
This is called the **global function** of j.

Transduction - MSPF

THEOREM 5 *Given a NOR-gate network η, let $\tilde{\eta}$ be the network obtained by replacing the node j by an OR gate. Let $\tilde{z}_k$ be the k-th primary output of $\tilde{\eta}$.*

$$f_j^M(\mathbf{x}) = \begin{cases} g_j(\mathbf{x}) & \textit{if } \exists k : \mathbf{x} \notin XDC_k \textit{ and } z_k(\mathbf{x}) \neq \tilde{z}_k(\mathbf{x}), \\ * & \textit{otherwise.} \end{cases}$$

$*$ *designates don't care points.*

Note: the OR gate just complements the output of f_j.

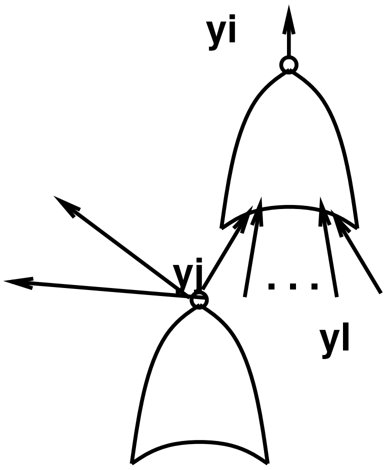
Expensive to compute $\tilde{z}_k$'s.

Can do better in case of no reconvergence.

MSPF for No Reconvergence

Define the MSPF for an edge $[j, i]$:

$$f^M_{[j,i]}(\mathbf{x}) = \begin{cases} 0 & \text{if } f^M_i(\mathbf{x}) = 1 \\ 1 & \text{if } f^M_i(\mathbf{x}) = 0 \text{ and } \forall l \in FI_i, l \neq j : g_l(\mathbf{x}) = 0 \\ * & \text{otherwise} \end{cases}$$



THEOREM 6 *Suppose* $\bigcap_{i \in FO_j} TFO_i = \phi$.

Then $f^M_j(\mathbf{x}) = \prod_{i \in FO_j} f^M_{[j,i]}(\mathbf{x})$.

Transduction - CSPF

- MSPF is expensive to compute.
- If the function of j is changed, f_i^M may change for some other node i.

Consider a set of functions $\{f_j^C\}$ for a set of nodes J s.t. a simultaneous replacement of the functions at all nodes $j \in J$ each by an arbitrary cover of f_j^C does not change the functionality of the network.

f_j^C is called a CSPF at j.

Note:

- CSPF's are defined for a set of nodes.
- We don't need to recompute CSPF's - they can be used independently.
- $f_j^C \subseteq f_j^M$

Transduction - CSPF

key ideas:

- Compute CSPF for one node at a time.
  - from the primary outputs to the primary inputs

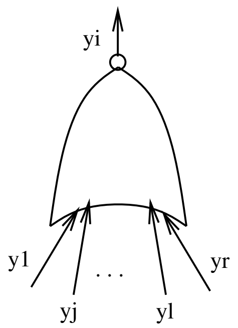
- If $(f_1^C, \ldots, f_{j-1}^C)$ have been computed, compute f_j^C so that a simultaneous replacement of the functions at the nodes $\leq j$ by those of $(f_1^C, \ldots, f_j^C)$ is valid.
  $\Rightarrow$ put more don't cares to those processed earlier.

- Compute CSPF's for edges so that
  $f_j^C(\mathbf{x}) = \prod_{i \in FO_j} f_{[j,i]}^C(\mathbf{x})$.

Note:
CSPF's are **dependent** on the orderings of nodes/edges.

CSPF's for Edges

CSPF f_i^C for a node i is given.



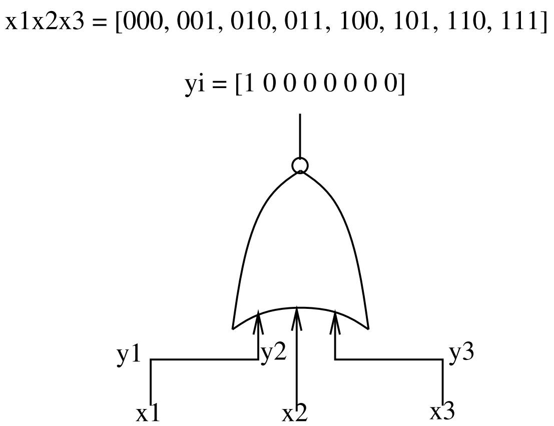
Ordering of Edges: $y_1 < y_2 < \cdots < y_r$
- put more don't cares to the edges processed earlier.

$$f_{[j,i]}^C(\mathbf{x}) = \begin{cases} 0 & \text{if } f_i^C(\mathbf{x}) = 1 \\ 1 & \text{if } f_i^C(\mathbf{x}) = 0 \text{ and} \\ & \forall y_l > y_j : g_l(\mathbf{x}) = 0 \text{ and } g_j(\mathbf{x}) = 1 \\ * & \text{otherwise} \end{cases}$$

Transduction - CSPF

Example:

$y_1 < y_2 < y_3$



$$
\begin{array}{lcl}
y_i & = & [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0] \\
\\
y_1 & = & [0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1] \\
y_2 & = & [0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1] \\
y_3 & = & [0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1] \\
f^C_([1,i]) & = & [0 \quad * \quad * \quad * \quad 1 \quad * \quad * \quad *] \\
f^C_([2,i]) & = & [0 \quad * \quad 1 \quad * \quad * \quad * \quad 1 \quad *] \\
f^C_([3,i]) & = & [0 \quad 1 \quad * \quad 1 \quad * \quad 1 \quad * \quad 1]
\end{array}
$$

Note: we just make the last 1 stay one and all others $*$.

Note:
CSPF for $[1, i]$ has the largest don't cares among the three edges.

CSPF Computation

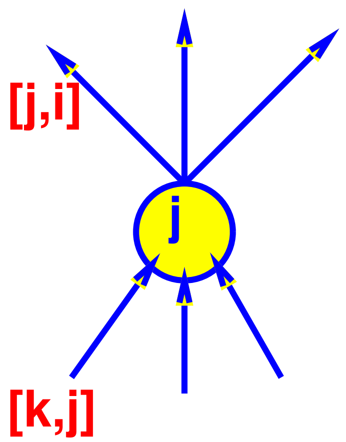
1. Compute the global functions g for all the nodes.

2. For each primary output z_k,

$$f^C_{z_k}(\mathbf{x}) = \begin{cases} * & \text{if } \mathbf{x} \in XDC_k, \\ g_{z_k}(\mathbf{x}) & \text{otherwise} \end{cases}$$

3. For each node j in a topological order from the outputs,

   - compute $f^C_j = \prod_{i \in FO_j} f^C_{[j,i]}$
   - compute CSPF for each fanin edge of j, i.e. $f^C_{[kj]}$.



Generalization of CSPF's

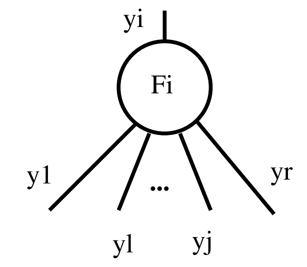
A Boolean network where a node function is arbitrary.

- Based on the same idea as Transduction
  - process one node at a time from the primary outputs.
  - compute a compatible don't care set for an edge, $f^C_{[j,i]}$.
  - intersect for all the fanout edges to compute a compatible don't care set for a node.

$$f^C_j = \prod_{i \in FO_j} f^C_{[j,i]}$$

Put more don't cares to nodes/edges processed earlier.

Compatible Don't Cares at Edges



Fi : node function at i

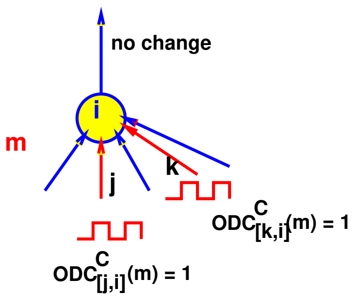
Ordering of edges: $y_1 < y_2 < \cdots < y_r$

- Assume no don't care at a node i.
- A compatible don't care for an edge $[j,i]$ is a function of FI_i: $ODC^C_{[j,i]} : B^r \rightarrow B$
  Note: it is a function of the local inputs. It is 1 at $m \in B^r$ if the input line [j,i] is a don't care.

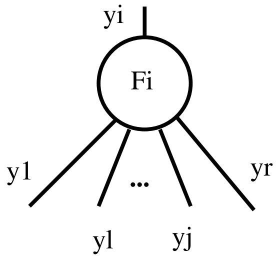
Property on $\{ODC^C_{[1,i]}, \ldots, ODC^C_{[r,i]}\}$:

For all $\mathbf{m} \in B^r$, the value of y_i does not change by arbitrarily flipping the values of

$\{y_j \in FI_i | ODC^C_{[j,i]}(\mathbf{m}) = 1\}$.



Compatible Don't Cares at Edges



Fi : node function at i

Given $\{ODC^C_{[1,i]}, \ldots, ODC^C_{[j-1,i]}\}$, compute $ODC^C_{[j,i]}$ s.t. $ODC^C_{[j,i]}(\mathbf{m}) = 1$ iff the insensitivity of the value of y_i with respect to y_j does not change by arbitrarily flipping the values of the y_k in the set:

$$\alpha = \{l \in FI_i | y_l < y_j, ODC^C_{[l,i]}(\mathbf{m}) = 1\}$$

Equivalently, $ODC^C_{[j,i]}(\mathbf{m}) = 1$ iff

$$\forall \tilde{\mathbf{m}}_\alpha \in B^{|\alpha|} : \overline{\frac{\partial F_i}{\partial y_j}}(\tilde{\mathbf{m}}_\alpha, \mathbf{m}_{\overline{\alpha}}) = 1$$

(Here $m = (m_\alpha, m_j, m_{\overline{\alpha}})$.) Thus we are arbitrarily flipping the m_α part. In some sense $m_{\overline{\alpha}}$ is enough to keep F_i insensitive to the value of y_j.

$\dfrac{\partial F_i}{\partial y_j} = (F_i \mid_{y_j=0}) \oplus (F_i \mid_{y_j=1})$
is called the Boolean difference of F_i with respect to y_j and is all conditions when F_i is sensitive to the value of y_j.
- this is a function of FI_i, but not y_j.

Compatible Don't Cares at a Node

Compatible don't care at a node j, ODC_j^C, is a function of the primary inputs.

- if j is a primary output, $ODC_j^C = XDC_j$
- otherwise,
  1. represent $ODC_{[j,i]}^C$ by the primary inputs for each fanout edge $[j,i]$.
  2. $ODC_j^C = \prod_{i \in FO_j} (ODC_i^C + ODC_{[j,i]}^C)$

THEOREM 7 *The set of incompletely specified functions with the ODC^C's computed above for all the nodes provides a set of CSPF's for an arbitrary Boolean network.*

Easier Computations of CODC Subset

On page 46 of the Savoj paper in the reader, an easy method for computing a CODC subset on each input of a function f is:

$$CODC_{y_k} = (\frac{\partial f}{\partial y_1} + \mathcal{C}_{y_1}) \ldots (\frac{\partial f}{\partial y_{k-1}} + \mathcal{C}_{y_{k-1}}) \overline{\frac{\partial f}{\partial y_k}} + CODC_f$$

where $CODC_f$ is the compatible don't care already computed for node f, and where f has its inputs $y_1, y_2, \ldots, y_r$ in that order. The notation $\mathcal{C}_y f = f_y f_{\overline{y}}$ is used here. The interpretation of the term for $CODC_{y_2}$

$$(\frac{\partial f}{\partial y_1} + \mathcal{C}_{y_1}) \overline{\frac{\partial f}{\partial y_2}}$$

is that: of the minterms $m \in B^r$ where f is insensitive to y_2, $(\overline{\frac{\partial f}{\partial y_2}})$, we allow m to be a don't care for y_2 if

1. either m is not a don't care for the y_1 input $(\frac{\partial f}{\partial y_1})$, or

2. if no matter what value is chosen for y_1, $(\mathcal{C}_{y_1})$, f is still insensitive to y_2 under m.

Generalization of CSPF's

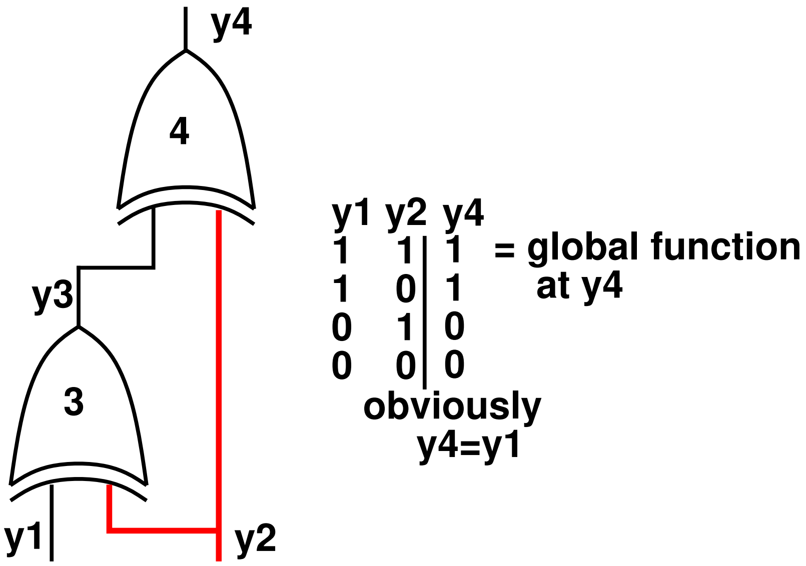
The proposed procedure have two drawbacks.

- ODC^C for an edge is expensive to compute. (exponential in # of fanins)
  - some heuristics have been proposed.
- The set of ODC_j^C's computed for the nodes is not maximal compatible, i.e.
  there might exist a function of the primary inputs for some node j s.t.
  1. the function cannot be obtained by ODC_j^C and
  2. the function can be added as a permissible function without destroying the property of the compatibility.

Non-Maximality of ODC^C

ODC^C computed at a node may not be maximally compatible.

Example:



$$ODC^C_{[2,3]} = ODC^C_{[2,4]} = \phi$$

$$ODC^C_2 = \phi$$

However, we can put either 1 or 0 at y_2.
$\Rightarrow$ ODC^C_2 should be $\equiv 1$.

Computing Don't Cares using BDD's

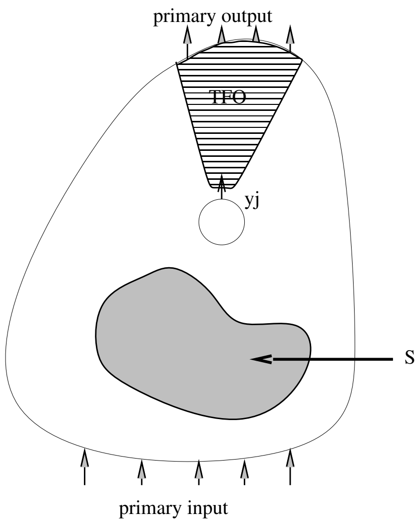
Handles arbitrary Boolean networks.

OUTLINE:

- MSPF's for general case
- CSPF's for general case
- The complete set of implementable functions for a given node
  - incorporating SDC's
- Maximally compatible set of implementable functions over all the nodes.

The Set of Implementable Functions

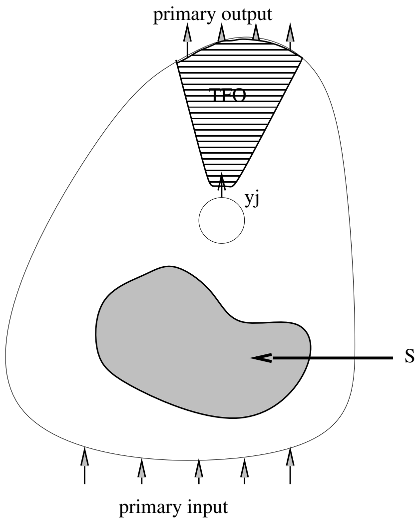
Problem:
Given a node j and a set of variables $S \subseteq \{x_1, \ldots, x_n, y_1, \ldots y_m\} - TFO_j$, find the set of implementable functions for j, where the variables of each function is a subset of S.



Note:

- SDC's are incorporated.
- $S = \{x_1, \ldots, x_n\} \Rightarrow$ MSPF
- $S = \{x_1, \ldots, x_n, y_1, \ldots y_m\} - TFO_j$
  $\Rightarrow$ the complete set of implementable functions at j

The Set of Implementable Functions



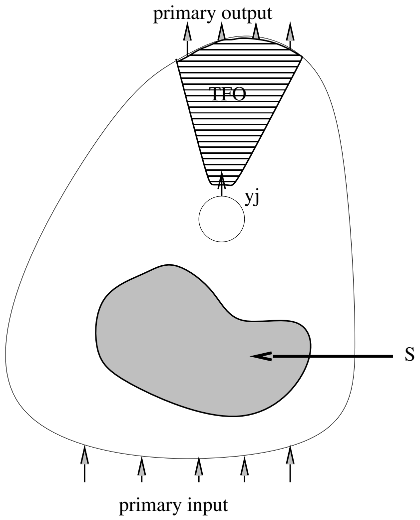
- $g_S : B^n \to B^{|S|}$ - the global function of S
  the i-th function of g_S is the global function of the i-th variable of S.

- $g_Z : B^n \to \{0, 1, *\}^p$ - the global function of the primary outputs $+XDC$

- $z^M : B^{n+1} \to B^p$ - the function of the primary outputs expressed by the primary inputs and y_j
  - for the node function of each primary output, recursively substitute each fanin by its node function, unless the fanin is either a primary input or y_j

The Maximal Set of Implementable Functions

Given S:



Consider a Boolean relation $F_j^M : B^{|S|} \times B \to B$ s.t. $F_j^M(\mathbf{s}, y_j) = 1$ iff

$$\forall \mathbf{x} \in B^n : (\mathbf{s} = g_S(\mathbf{x})) \Rightarrow z^M(\mathbf{x}, y_j) \in g_Z(\mathbf{x})$$

Note: F_j^M represents an incompletely specified function, i.e. if an s is associated with both $y_j = 1$ and $y_j = 0$ then $y_j = *$ (don't care for that s).

THEOREM 8 *The set of functions given by F_j^M is the set of implementable functions at j expressed by S.*

Proof. Easy.

BDD Computation for F_j^M

$F_j^M(s, y_j)$ is a relation - a one to many map from $s \rightarrow y_j$.

$F_j^M(\mathbf{s}, y_j) = 1$ iff

$$\forall \mathbf{x} \in B^n : \mathbf{s} = g_S(\mathbf{x}) \Rightarrow z^M(\mathbf{x}, y_j) \in g_Z(\mathbf{x})$$

Universal Quantification
For a function $h : B^r \rightarrow B$ of variables $V = \{v_1, \ldots, v_r\}$ and $V_k = \{v_1, \ldots, v_k\} \subseteq V$, the **universal quantification** of h by V_k is the function h_k of the variables $V - V_k$ s.t. $h_k(\mathbf{v}_{r-k}) = 1$ iff $\forall \mathbf{v}_k \in B^k : h(\mathbf{v}_k, \mathbf{v}_{r-k}) = 1$.

- h_k is computed by successive cofactors of h by V_k.

  $$h_k(\mathbf{v}_{r-k}) = C_{v_1}(C_{v_2}(\cdots C_{v_k}(h)\cdots))$$

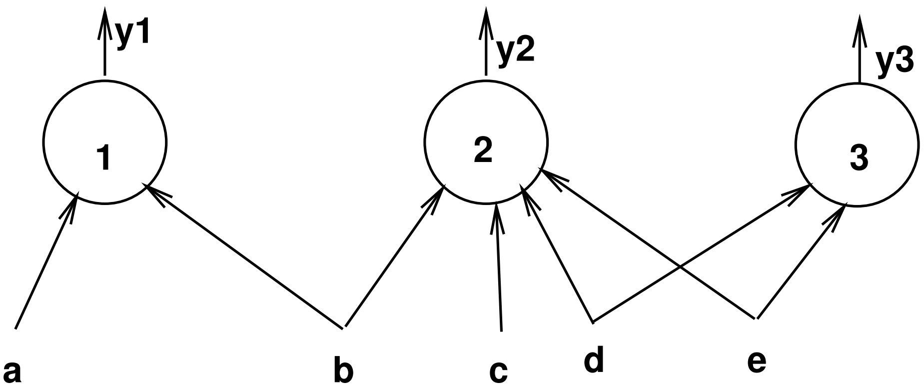
  where $C_{v_i}(h) = h \mid_{v_i=0} h \mid_{v_i=1}$

  The computation can be done easily using BDD's.

- Thus F_j^M can be computed and represented using BDD's.

The Set of Implementable Functions

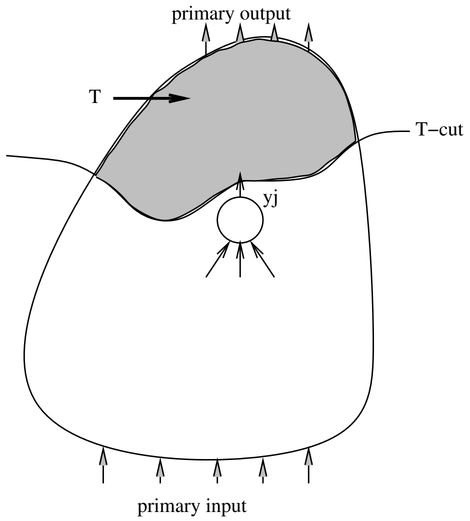
Remark:
The BDD for F_j^M may be large.
Typically, S is set to FI_j plus those variables whose fanins are a subset of FI_j.



In processing y_2 we use y_3 since its support is a subset of y_2.

This is called the support filter.
This corresponds to Boolean resubstitution since F_j^M is a function of y_3.

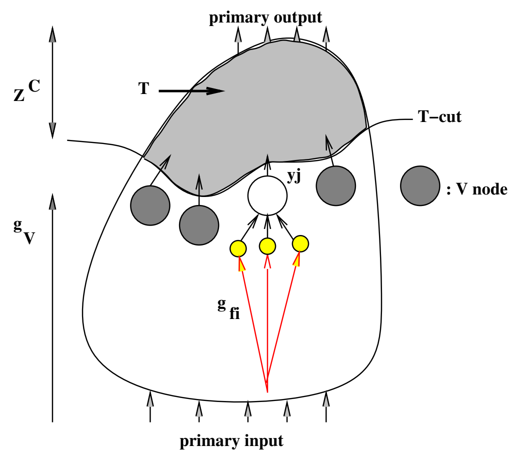
Compatible Set of Functions



- T: a set of nodes s.t. $TFO_j - \{j\} \subseteq T$ and $T \cap (TFI_j \cup \{j\}) = \phi$.
  - suppose a set of functions $\{f_i^C | i \in T\}$ is given. f_i^C is an incompletely specified function of FI_i
- find f_j^C compatible with $\{f_i^C | i \in T\}$
  - f_j^C is a maximal set of functions of FI_j compatible with all the combinations of the completely specified functions given by $\{f_i^C | i \in T\}$.

Note: f_j^C is expressed by its fanins, rather than the primary inputs.

Compatible Set of Functions



- V: the set of nodes except j adjacent to T-cut
  $V = \{l \mid l \notin T,\ l \neq j, \text{ and } \exists i \in T : i \in FO_l\}$

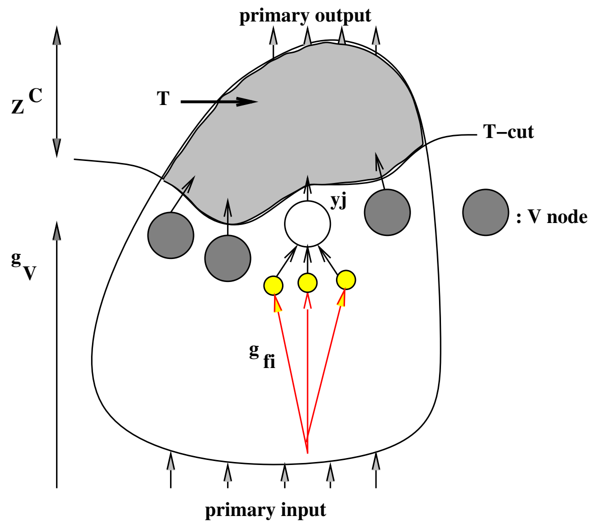
- $g_V : B^n \rightarrow B^{|V|}$ - the global functions at the V nodes.

- $g_{\mathsf{fi}} : B^n \rightarrow B^{|FI_j|}$ - the global functions at the FI_j nodes.

- $Z^C : B^{|V|+1} \times B^{|Z|} \rightarrow B$ - a relation on the functionality of T (relation between $V \cup \{y_j\}$ and the primary outputs)

  – $Z^C((\mathbf{v}, y_j), \mathbf{z}) = 1$ iff
    there exists a set of completely specified functions at T given by $\{f_i^C | i \in T\}$ with which $\mathbf{z}$ appears at the primary outputs by setting $(\mathbf{v}, y_j)$ at V and y_j.

Compatible Set of Functions



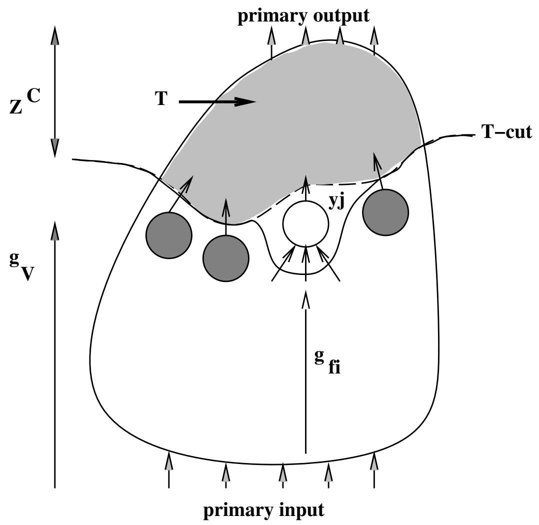
Consider a Boolean relation $F_j^C : B^{|FI_j|} \times B \to B$ s.t. $F_j^C(\mathbf{y}, y_j) = 1$ iff

$$\forall (\mathbf{x}, \mathbf{z}) \in B^n \times B^p : \quad \begin{array}{l} (1) \quad \mathbf{y} = g_{\mathsf{fi}}(\mathbf{x}) \text{ and} \\ (2) \quad Z^C((g_V(\mathbf{x}), y_j), \mathbf{z}) = 1 \\ \Rightarrow \mathbf{z} \in g_Z(\mathbf{x}) \end{array}$$

Note: $F_j^C(\mathbf{y}, y_j)$ gives a relation (incompletely specified function) between $\mathbf{y}$ and y_j.

THEOREM 9 *F_j^C is a maximal set of functions of FI_j compatible with $\{f_i^C | i \in T\}$.*

Compatible Set of Functions



Once F_j^C is computed,

- T-cut is updated, and y_j is put in T.
  The nodes in FI_j are now adjacent to the cut.

- Z^C is updated.
  The new Z^C is a relation between $V \cup FI_j$ and the primary outputs.
  The new $Z^C((\mathbf{v}, \mathbf{y}), \mathbf{z}) = 1$ iff

$$\exists y_j : Z^C((\mathbf{v}, y_j), \mathbf{z}) = 1 \text{ and } F_j^C(\mathbf{y}, y_j) = 1$$

BDD Computation for F_j^C and Z^C

$F_j^C(\mathbf{y}, y_j) = 1$ iff

$$\forall(\mathbf{x}, \mathbf{z}) \in B^n \times B^p : \begin{array}{l} (1) \quad \mathbf{y} = g_{\mathsf{fi}}(\mathbf{x}) \text{ and} \\ (2) \quad Z^C((g_V(\mathbf{x}), y_j), \mathbf{z}) = 1 \\ \Rightarrow \mathbf{z} \in g_Z(\mathbf{x}) \end{array}$$

The new $Z^C((\mathbf{v}, \mathbf{y}), \mathbf{z}) = 1$ iff

$$\exists y_j : Z^C((\mathbf{v}, y_j), \mathbf{z}) = 1 \text{ and } F_j^C(\mathbf{y}, y_j) = 1$$

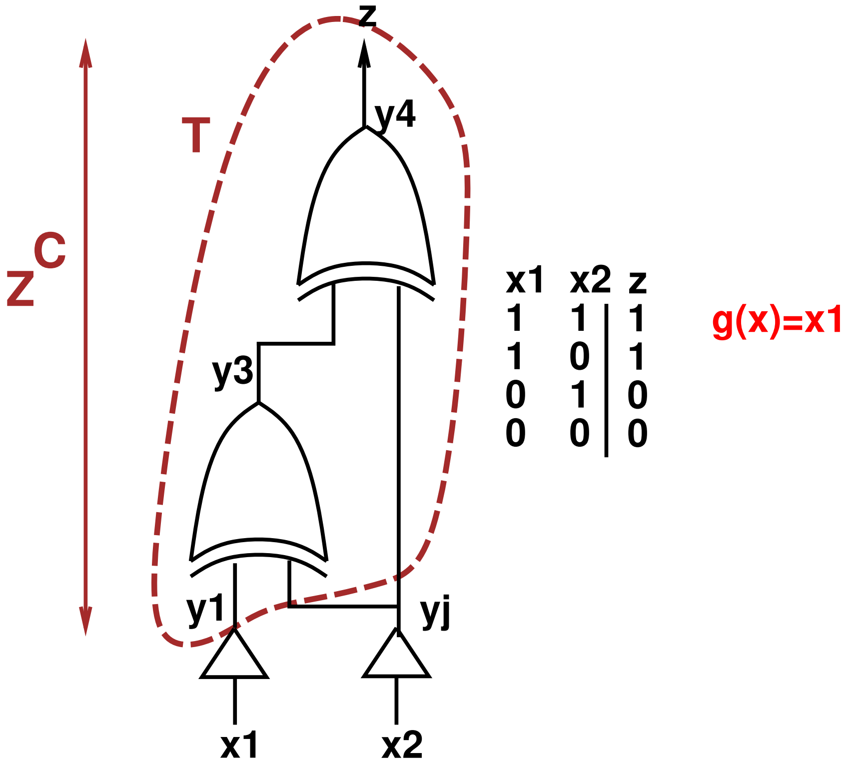
Existential Quantification
For a function $h : B^r \to B$ of variables $V = \{v_1, \ldots, v_r\}$ and $V_k = \{v_1, \ldots, v_k\} \subseteq V$, the **existential quantification** of h by V_k is the function h_k of the variables $V - V_k$ s.t. $h_k(\mathbf{v}_{r-k}) = 1$ iff $\exists \mathbf{v}_k \in B^k : h(\mathbf{v}_k, \mathbf{v}_{r-k}) = 1$.

$$h_k(\mathbf{v}_{r-k}) = S_{v_1}(S_{v_2}(\cdots S_{v_k}(h) \cdots))$$

where $S_{v_i}(h) = h \mid_{v_i=0} + h \mid_{v_i=1}$.

Compatible Set of Functions

Example:



$$
\begin{array}{ll}
T = \{y_4, y_3\}, & V = \{y_1\} \\
f_4^C = y_j \oplus y_3, & f_3^C = y_1 \oplus y_j \\
g_V = x_1, & g_{\mathsf{fi}} = x_2 \\
g_Z = x_1, & Z^C = y_1 \overline{\oplus} z
\end{array}
$$

Thus

$$
\begin{aligned}
F_j^C(x_2, y_j) &= \forall_{\{x_1, z\}} \\
&\quad ((x_2 = g_{fi}) \wedge (Z^C((g_V(x), y_j), z)) \Rightarrow (z = g_Z(x \\
&= C_{\{x_1, z\}}((x_2 \overline{\oplus} g_{fi})(x_1 \oplus z) \Rightarrow (z \overline{\oplus} x_1)) \\
&= 1
\end{aligned}
$$

Summary of Computation of Don't Cares

- XDC is given, SDC is easy to compute.
- Transduction
  - NOR-gate networks
  - MSPF, CSPF (permissible functions of PI)
- Extension of CSPF's for general networks
  - expensive to compute, not maximal
  - implementable functions of PI and y.
- Direct computation using BDD's
  - the complete set of implementable functions
  - maximally compatible set of implementable functions expresssed by fanins
  - SDC's are also incorporated.
- Subset of don't cares
  - TFI_j and the nodes whose support are a subset of the fanins of j.

Minimizing Local Functions

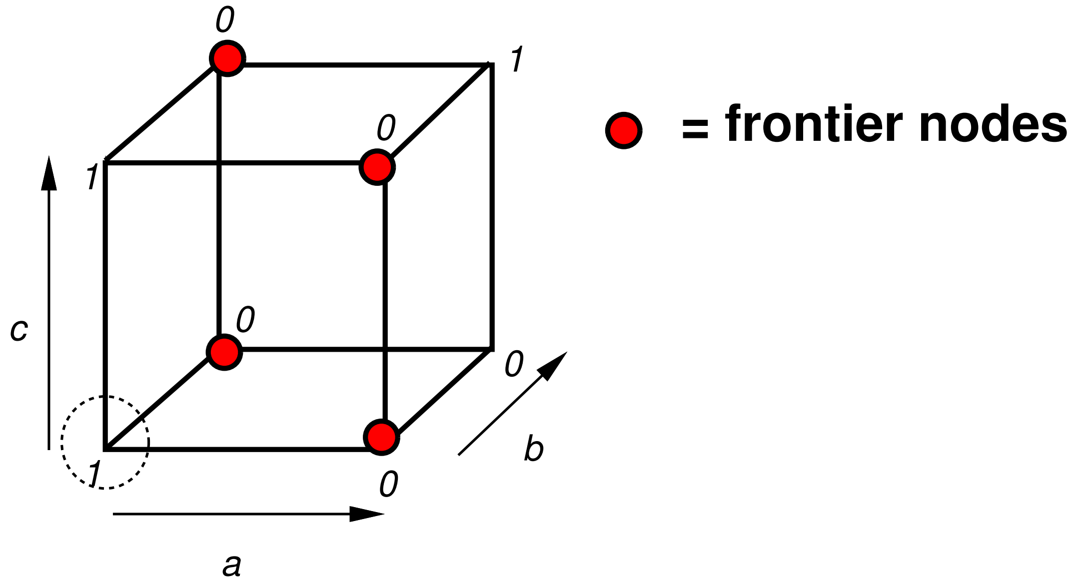
Once an incompletely specified function is derived, we minimize it.

1. Minimize the number of literals
   - Minimum_literal() in P.16-21 in the slides for Boolean division
2. The off-set of the function may be too large.
   - Reduced offset minimization
   - Tautology-based minimization
   - ATPG techniques

Reduced Offset

Idea: In expanding any single cube, only part of the off-set is useful. This part is called the **reduced offset**

Example:



In expanding the cube $\overline{a}\overline{b}\overline{c}$ the point abc is of no use. Therefore during this expansion it might as well be put in the offset. Then the offset which was $a\overline{b}+a\overline{c}+\overline{a}b+b\overline{c}$ becomes $a+b$.

Tautology-based Two-Level Minimization

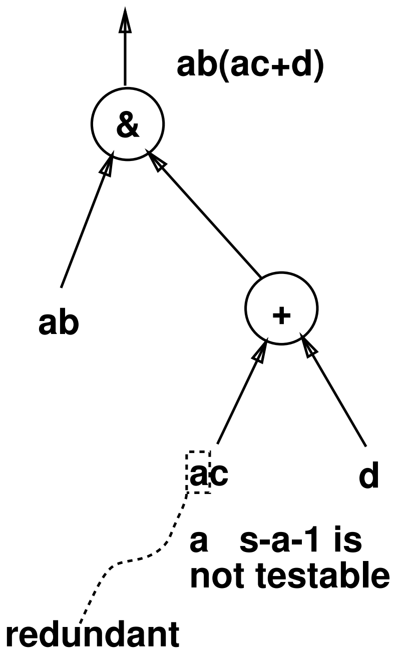
Two-level minimization can be done without using the offset. Offset is used for blocking expansion of a cube too far. The other method of expansion is based on tautology.

Example:
In expanding a cube $c = a\overline{b}e\overline{g}$ to $ae\overline{g}$ we can test if $ae\overline{g} \subseteq f + d$. This can be done by testing $(f + d)_{ae\overline{g}} \equiv 1$ i.e. is $(f + d)_{ae\overline{g}}$ a tautology?

ATPG

Multi-Level Tautology:

Idea:
Use testing methods. (ATPG is a highly developed technology). If we can prove that a fault is untestable, then the untestable signal can be set to a constant.



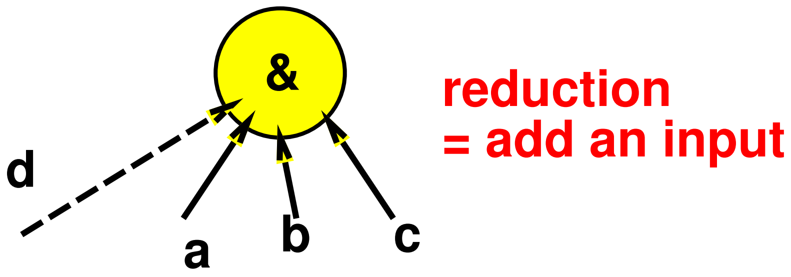
This is like **expansion** in espresso.

Removing Redundancies

Redundancy removal:

- do random testing to locate easy tests
- apply new ATPG methods to prove testability of remaining faults.

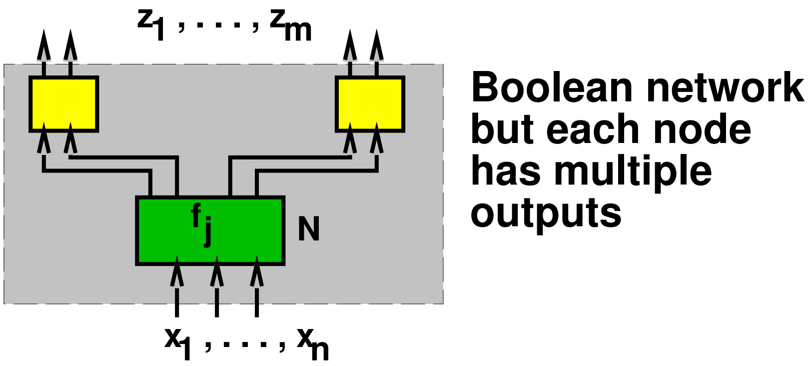
Reduction is the reverse of this. It is done to make something else untestable. There is a close analogy with reduction in espresso. Here also reduction is used to move away from a local minimum.



used in transduction (Muroga)
used in global flow (Bermen and Trevellan)

Multiple Output Functions

Network of Multiple Output Functions:

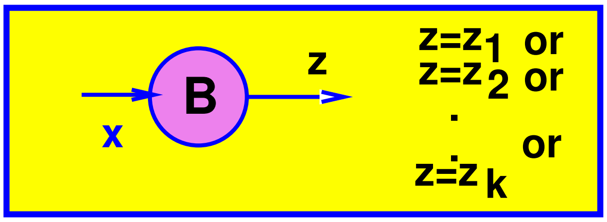


What is the set of implementable functions for f_j at N?

- Don't care formulation does not work
- **Boolean Relation:** (completely specified)

$$\mathcal{B} : B^n \longrightarrow 2^{B^m}$$

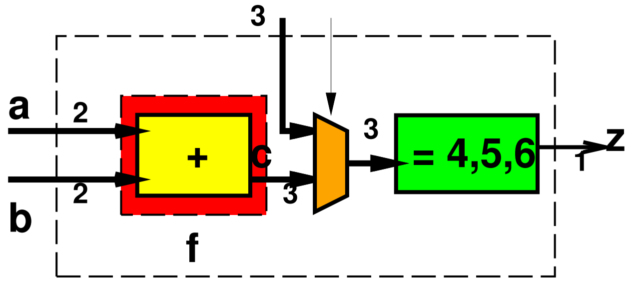
(one to many mapping)



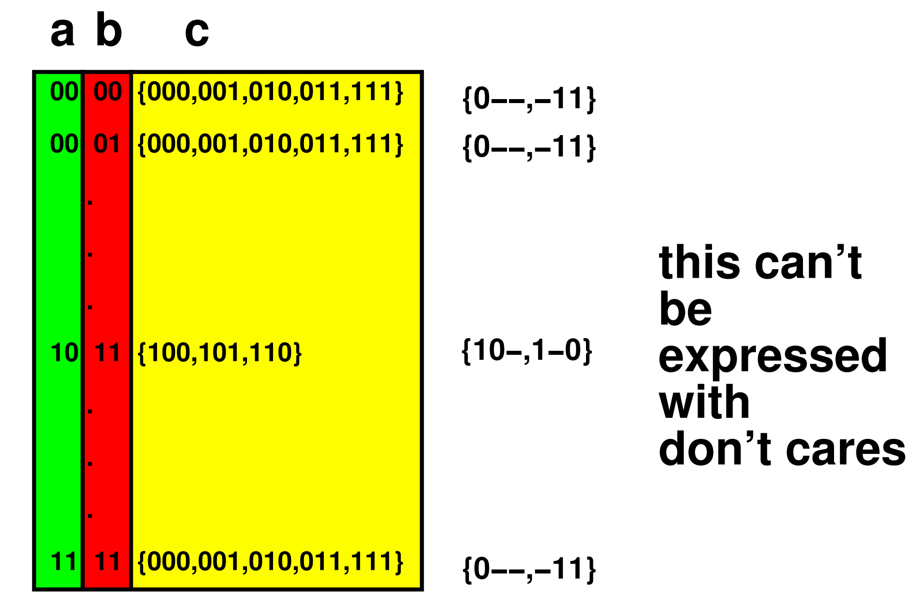
- **Implementable functions:** A function g_j is an implementation (cover) of a Boolean relation $\mathcal{B}$ (or contained in):

$$g_j \prec \mathcal{B} \Longleftrightarrow \forall x, g_j(x) \in \mathcal{B}(x)$$

Example: Boolean Relation



Adder followed by comparator



Boolean relation for c

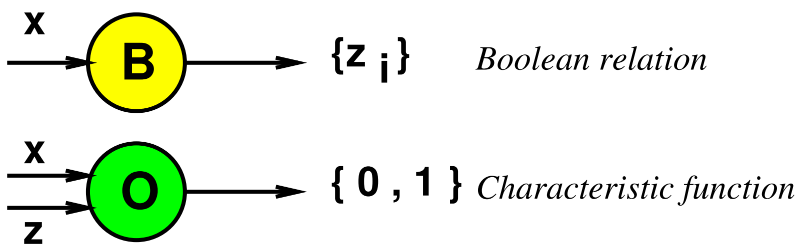
Want: $f \prec \mathcal{B}$
One Solution:

$$
\begin{aligned}
f &= 000 \text{ if } a + b \neq 4,5,6 \\
&= 100 \text{ if } a + b = 4,5,6
\end{aligned}
$$

Representing Boolean Relations

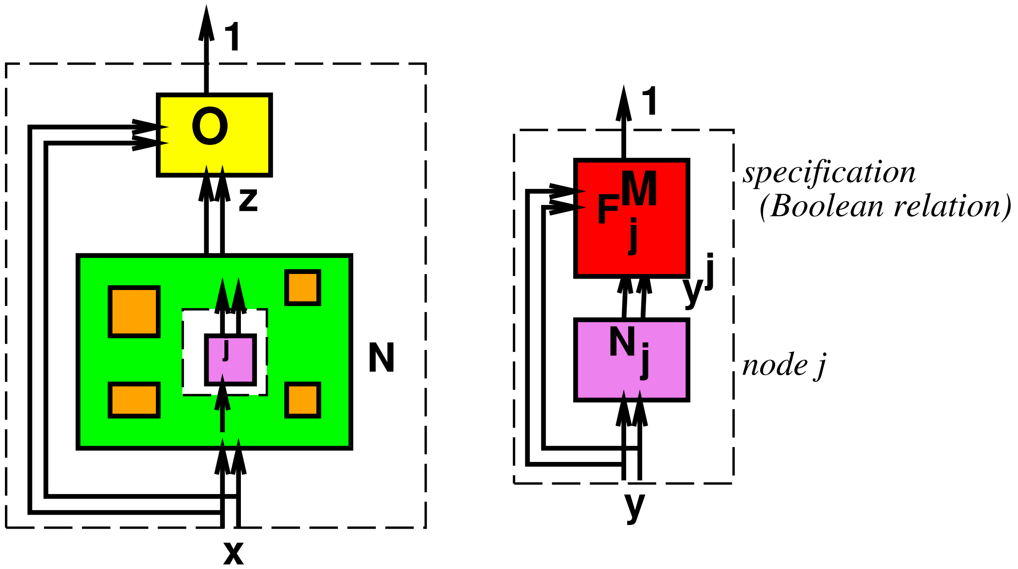
Observability Function:

$\mathcal{O}$ = characteristic function of Boolean relation $\mathcal{B}$.



$$\mathcal{O}(x, z) = 1 \iff z \in \mathcal{B}(x)$$

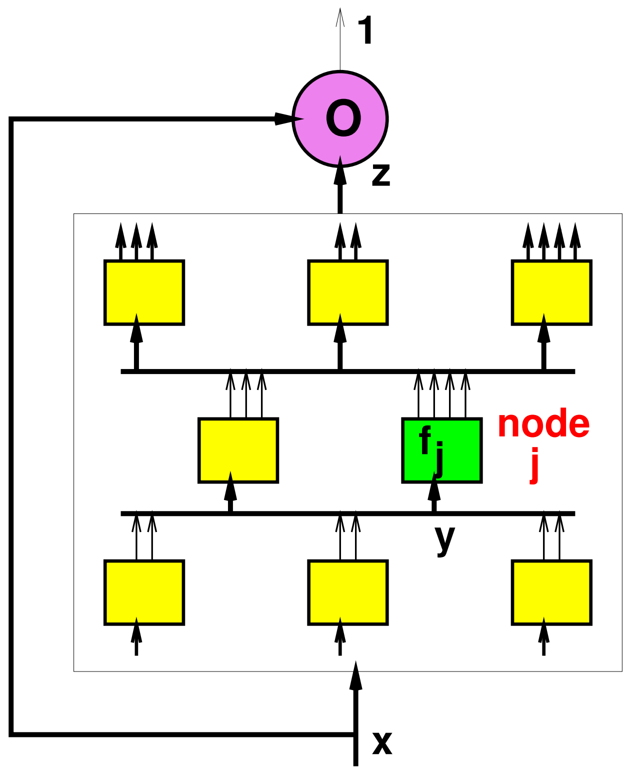
Observability Network:



- network of multiple output nodes
  - *same computational model*
  - *hierarchical formulation*

General Problem

Given network of multiple-output functions and Observability specification:



Summarize all implementable functions at node j, i.e. derive $F_j^M(\mathbf{y}, \mathbf{y}^j)$ such that:

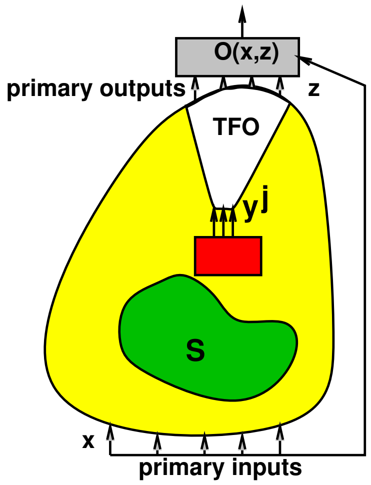
f is implementable at node $j \iff f \prec F_j^M(\mathbf{y}, \mathbf{y}^j)$

{ *Cerny - 1976, Savoj Thesis - 1992, IWLS'93* }

Optimization Problem: Find "best f_j : $f_j \prec F_j^M(\mathbf{y}, \mathbf{y}^j)$

Deriving the Maximum Flexibility

Given S:



Consider a Boolean relation $F_j^M : B^{|S|} \times B^k \rightarrow B$ s.t.

$$F_j^M(\mathbf{s}, \mathbf{y}^j) = \forall \mathbf{x}[(\mathbf{s} = g_S(\mathbf{x})) \Rightarrow \forall \mathbf{z}[(\mathbf{z} = Z^M(\mathbf{x}, \mathbf{y}^j)) \Rightarrow \mathcal{O}(\mathbf{x}, \mathbf{z})]]$$

Note: $Z^M(\mathbf{x}, \mathbf{y}^j)$ represents the mapping given by the current circuit between $\mathbf{x}, \mathbf{y}^j$ and the outputs $\mathbf{z}$.

Theorem: *The set of functions given by $f \prec F_j^M$ is the set of* **all** *implementable multi-output functions at node j expressed as functions whose support is in S.*

Note: $\mathcal{O}$ can also be treated just as any other node in the Boolean network. This means that even if we start with an initial Boolean relation specification, and a network of single output nodes, we can use the standard methods (*e.g. full_simplify*) for optimization.

Boolean Relation Minimizers:

- Exact - Somenzi (1989),

  Like Quine-McCluskey

  - define prime implicants
  - generate all primes
  - solve covering (binate) problem

- Heuristic - Watanabe (1991)

  Uses BDD's to represent Boolean relation - iterates reduce, expand irredundant until no change.