Logic Synthesis and Verification

Jie-Hong Roland Jiang
江介宏

Department of Electrical Engineering
National Taiwan University

Fall 2024

Don't Cares and Node Minimization

Reading:

Logic Synthesis in a Nutshell

Section 3 (§3.4)

part of the following slides are by
courtesy of Andreas Kuehlmann

Node Minimization

Problem:

- Given a Boolean network, optimize it by minimizing each node as much as possible

Note:

- Assume initial network structure is given
  - Typically obtained after the global optimization, e.g. division and resubstitution
- We minimize the function associated with each node

Permissible Functions of a Node

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

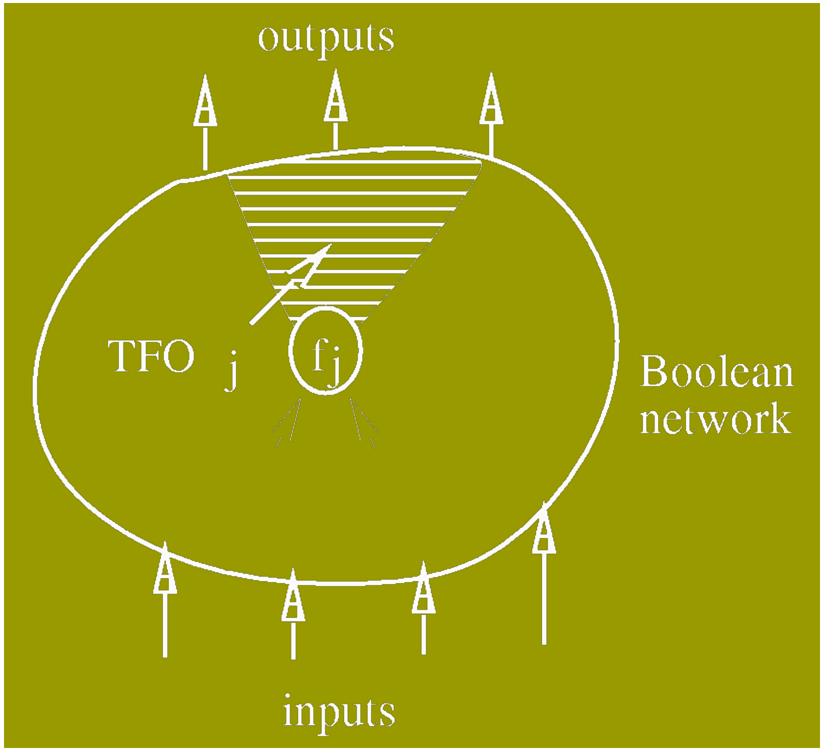
Definition:

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

- the variables of g_j do not intersect with TFO_j
  - $TFO_j = \{\text{node } i: i = j \text{ or } \exists \text{ path from } j \text{ to } i\}$
- the replacement of the function associated with j, say f_j, by g_j does not change the functionality of the network

Permissible Functions of a Node

- The set of implementable (or permissible) functions at j provides the solution space of the local optimization at node j



TFOj = {node i: i = j or ∃ path from j to i}

Prime and Irredundant Boolean Network

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

- Definition: F_j is prime (in a multilevel sense) if for all cubes $c \in F_j$, no literal of c can be removed without changing the functionality of the network

- Definition: F_j is irredundant if for any cube $c \in F_j$, the removal of c from F_j changes the functionality of the network

- Definition: A Boolean network is prime and irredundant if F_j is prime and irredundant for all j

Node Minimization

Goals:

- Given a Boolean network:
  1. make the network prime and irredundant
  2. for a given node of the network, find a least-cost SOP expression among the implementable functions at the node

Note:

- Goal 2 implies Goal 1
- There are many expressions that are prime and irredundant, just like in two-level minimization. We seek the best.

Taxonomy of Don't Cares

- External don't cares - XDC
  - The set of don't care minterms (in terms of primary input variables) given for each primary output is denoted XDC_k, $k=1,\ldots,p$

- Internal don't cares derived from the network structure
  - Satisfiability don't cares SDC
  - Observability don't cares ODC

- Complete Flexibility - CF
  - CF is a superset of SDC, ODC, and localized XDC

Satisfiability Don't Cares

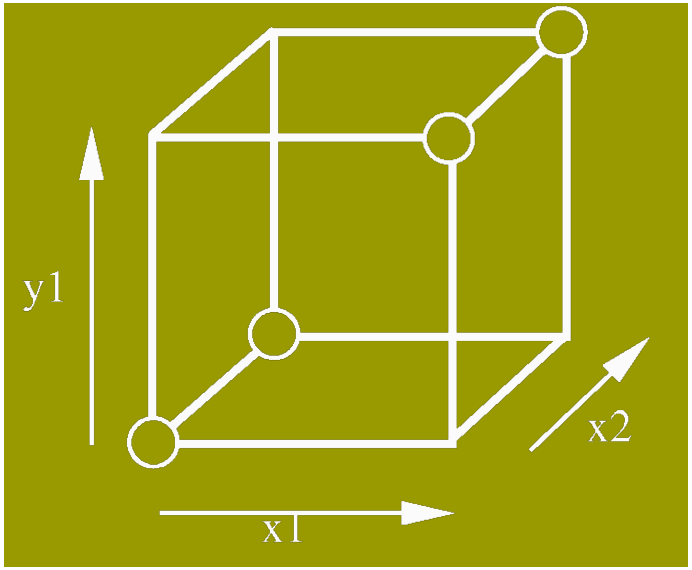
- We may represent a node using the *n* primary inputs plus the *m* intermediate variables
  - Boolean space is B^{n+m}
- However, intermediate variables depend on the primary inputs
- Thus not all the minterms of B^{n+m} can occur:
  - use the non-occurring minterms as don't cares to optimize the node function
  - we get internal don't cares even when no external don't cares exist

Satisfiability Don't Cares

- Example

  $y_1 = F_1 = \neg x_1$
  $y_j = F_j = y_1 x_2$

  - Since $y_1 = \neg x_1$, $y_1 \oplus \neg x_1$ never occurs. So we may include these points to represent F_j
    $\Rightarrow$ Don't Cares
  - *SDC* $= (y_1 \oplus \neg x_1)+(y_j \oplus y_1 x_2)$



In general,

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

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

Observability Don't Cares

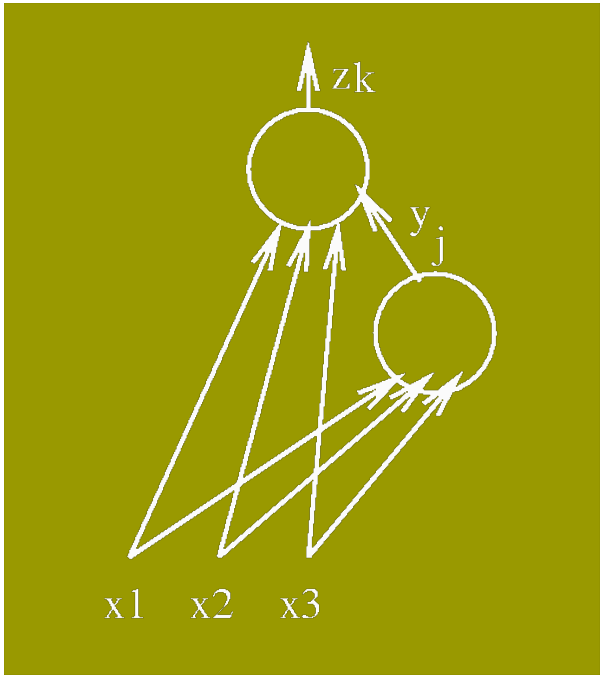
$y_j = \neg x_1 x_2 + x_1 \neg x_3$
$z_k = x_1 x_2 + y_j \neg x_2 + \neg y_j \neg x_3$

- Any minterm of
  $x_1 x_2 + \neg x_2 \neg x_3 + x_2 x_3 + x_1 \neg x_3$
  determines z_k independent of y_j
- The ODC of y_j w.r.t. z_k is the set of minterms of the primary inputs for which the value of y_j is not observable at z_k

$$ODC_{jk} = \{x \in B^n \mid z_k(x)|_{y_j=0} \equiv z_k(x)|_{y_j=1}\}$$

  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 value for z_k when $x \in ODC_{jk}$
- The ODC of y_j w.r.t. all primary outputs is $ODC_j = \cap_k ODC_{jk}$

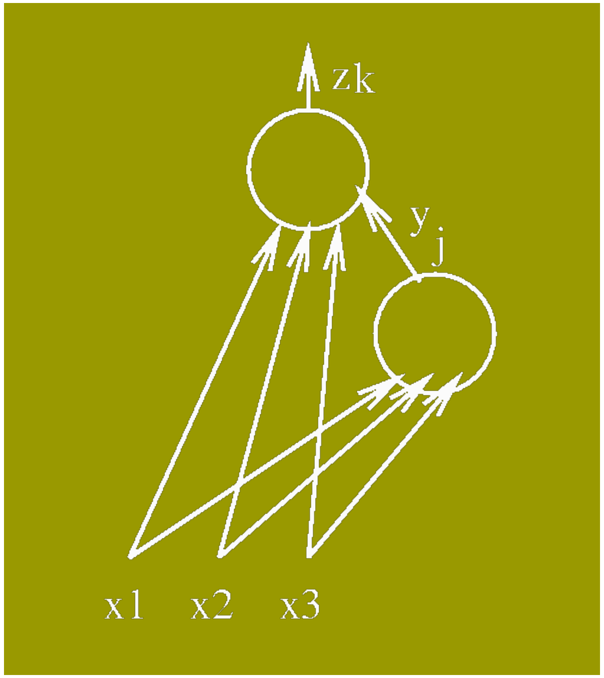


Observability Don't Cares

$$ODC_{jk} = \{x \in B^n \mid z_k(x)|_{y_j=0} = z_k(x)|_{y_j=1}\}$$

denote $ODC_{jk} = \overline{\dfrac{\partial z_k}{\partial y_j}}$

where $\dfrac{\partial z_k}{\partial y_j} = z_k(x)|_{y_j=0} \oplus z_k(x)|_{y_j=1}\}$



Observability Don't Cares

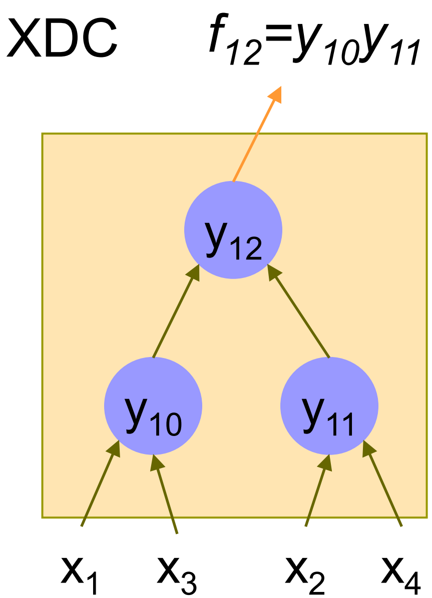
- The ODCs of node i and node j in a Boolean network may not be compatible
  - Modifying the function of node i using ODC_i may invalidate ODC_j
  - It brings up the issue of compatibility ODC (CODC)
  - Computing CODC is too expensive to be practical
    - Practical approaches to node minimization often consider one node at a time rather than multiple nodes simultaneously

External Don't Cares

- The XDC global for an entire Boolean network is often given
- The XDC local for a specified window in a Boolean network can be computed
- Question:
  - How do we represent XDC?
  - How do we translate XDC into local don't care?
    - XDC is originally in PI variables
    - Translate XDC in terms of input variables of a node

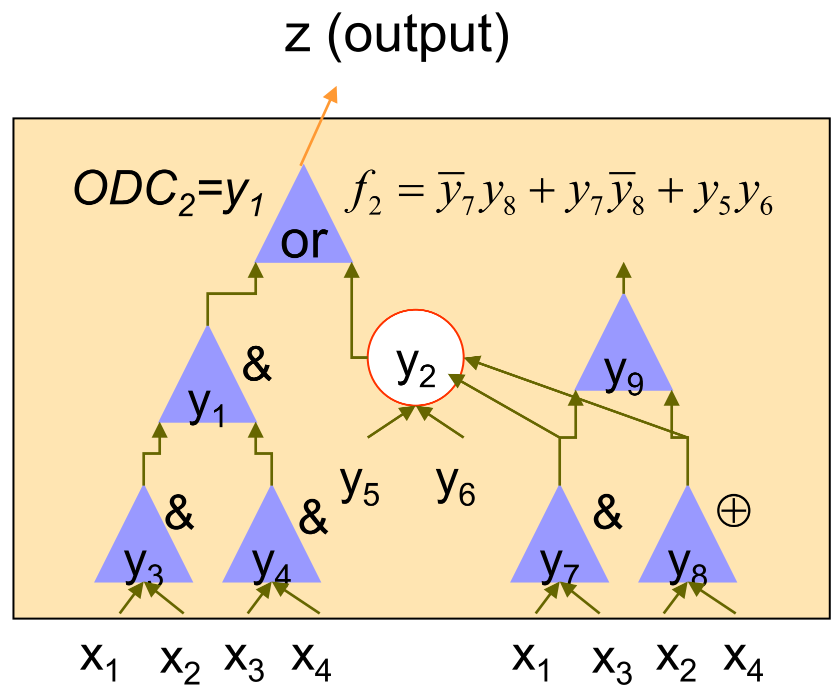
External Don't Cares

- Representing XDC



XDC in a separate network

$y_{10} = x_1 x_3 \qquad y_{11} = \overline{x}_2 \overline{x}_4$

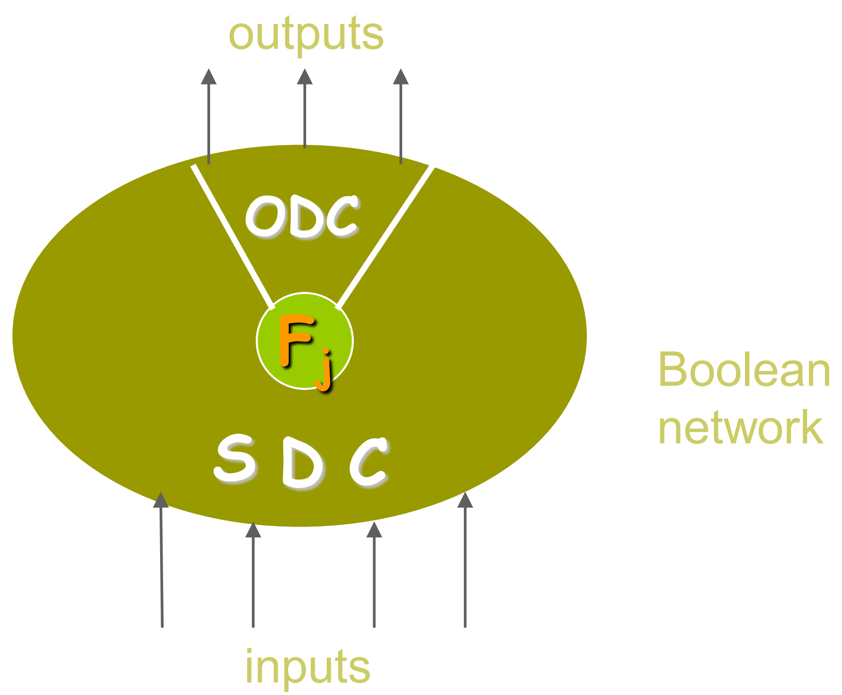


multi-level Boolean network for z

Don't Cares of a Node

- The don't cares of a node j can be computed by

$$DC_j = \sum_{i \notin TFO_j} (y_i \overline{F}_i + \overline{y}_i F_i) + \prod_{k=1}^{p} (ODC_{jk} + XDC_k)$$

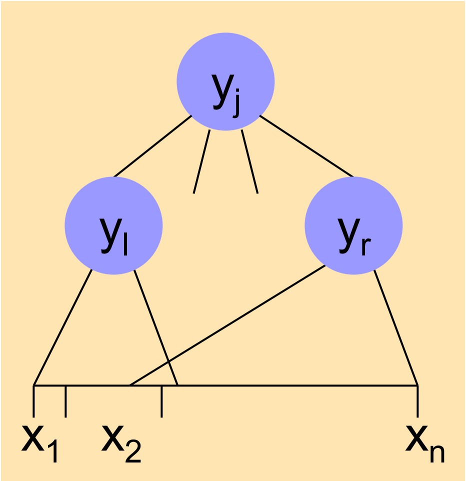


Don't Cares of a Node

- Theorem: The function $\mathcal{F}_j$ = (F_j-DC_j, DC_j, ¬(F_j+DC_j)) is the complete set of implementable functions at node j
- Corollary: F_j is prime and irredundant (in the multi-level sense) iff it is prime and irredundant cover of $\mathcal{F}_j$
- A least cost expression at node j can be obtained by minimizing $\mathcal{F}_j$
- A prime and irredundant Boolean network can be obtained by using only 2level logic minimization for each node j with the don't care DC_j

Mapping Don't Cares to Local Space

- How can ODC + XDC be used for optimizing a node j?
  - ODC and XDC are in terms of the primary input variables
    - Need to convert to the input variables of node j



Mapping Don't Cares to Local Space

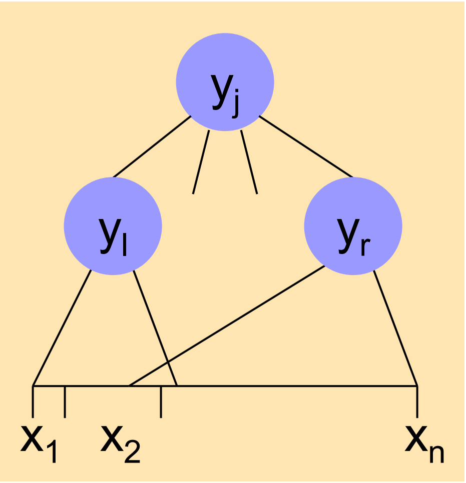
- **Definition:** The local space B^r of node j is the Boolean space spanned by the fanin variables of node j (plus maybe some other variables chosen selectively)
  - A don't care set $D(y^{r+})$ computed in local space spanned by y^{r+} is called a local don't care set. (The "+" stands for additional variables.)
  - **Solution:** Map DC(x) = ODC(x) + XDC(x) to local space of the node to find local don't cares, i.e.,

$$D(y^{r+}) = \overline{IMG_{g_{FI_j^+}}(\overline{DC}(x))}$$

Mapping Don't Cares to Local Space

- Computation in two steps:
  1. Find DC(x) in terms of primary inputs
  2. Find D, the local don't care set, by image computation and complementation

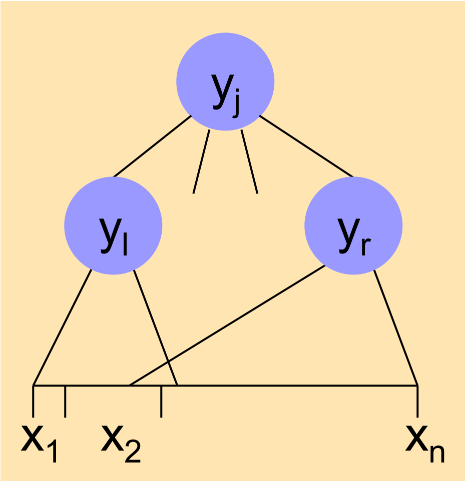
$$D(y^{r+}) = \overline{IMG_{g_{FI_j^+}}(\overline{DC}(x))}$$



Mapping Don't Cares to Local Space
Global Function of a Node

$$y_j = \begin{cases} f_j(y_k, \cdots, y_l) \\ g_j(x_1, \cdots, x_n) & \text{global function} \end{cases}$$

$$B^{m+n} \to B^n$$



Mapping Don't Cares to Local Space Don't Cares in Primary Inputs

- BDD based computation
  - Build BDD's representing global functions at each node
    - in both the primary network and the don't care network, $g_j(x_1,\ldots,x_n)$
    - use BDD_compose
  - Replace all the intermediate variables in (ODC+XDC) with their global BDDs

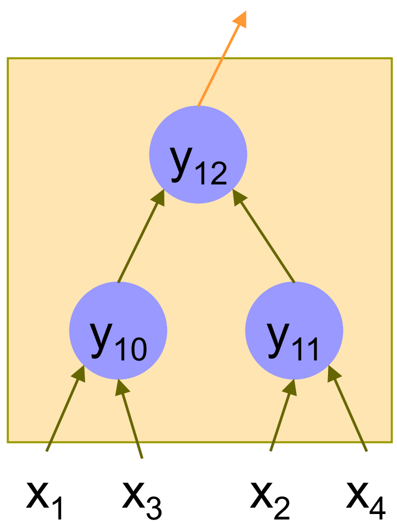
$$\tilde{h}(x,y) = DC(x,y) \rightarrow h(x) = DC(x)$$

$$\tilde{h}(x,y) = \tilde{h}(x,g(x)) = h(x)$$

Mapping Don't Cares to Local Space

- Example

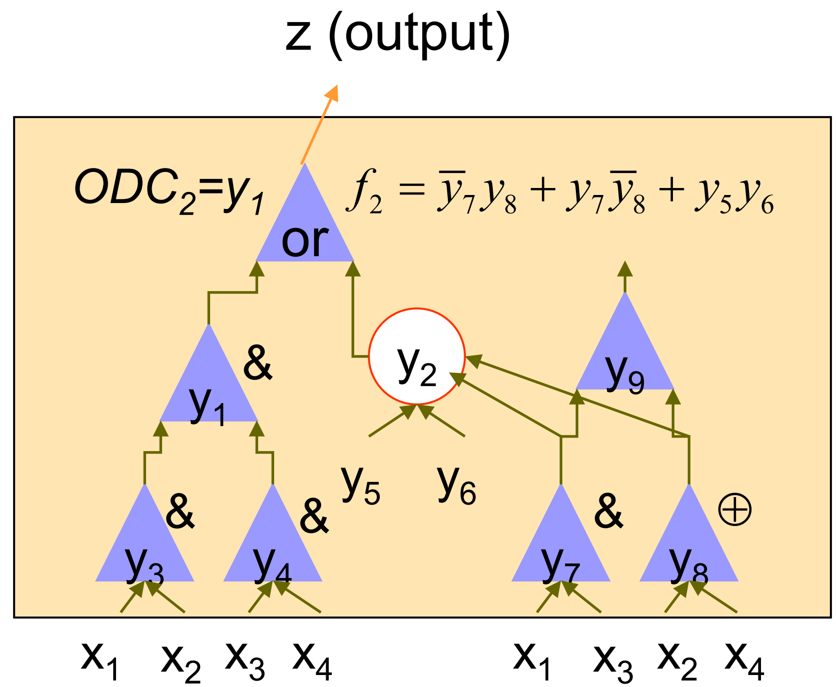
XDC $f_{12}=y_{10}y_{11}$



$y_{10} = x_1 x_3 \quad y_{11} = \overline{x}_2 \overline{x}_4$

$XDC_2 = y_{12}$

$g_{12} = x_1 \overline{x}_2 x_3 \overline{x}_4$



$ODC_2 = y_1$

$g_1 = x_1 x_2 x_3 x_4$

$DC_2 = ODC_2 + XDC_z$

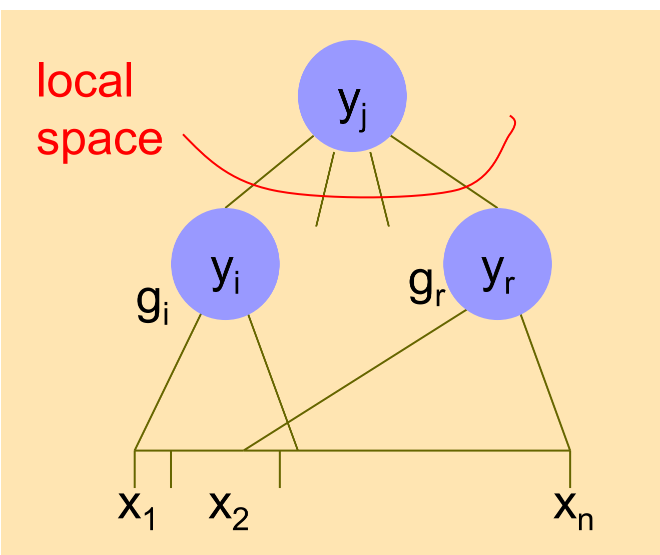
$DC_2 = x_1 x_2 x_3 x_4 + x_1 \overline{x}_2 x_3 \overline{x}_4$

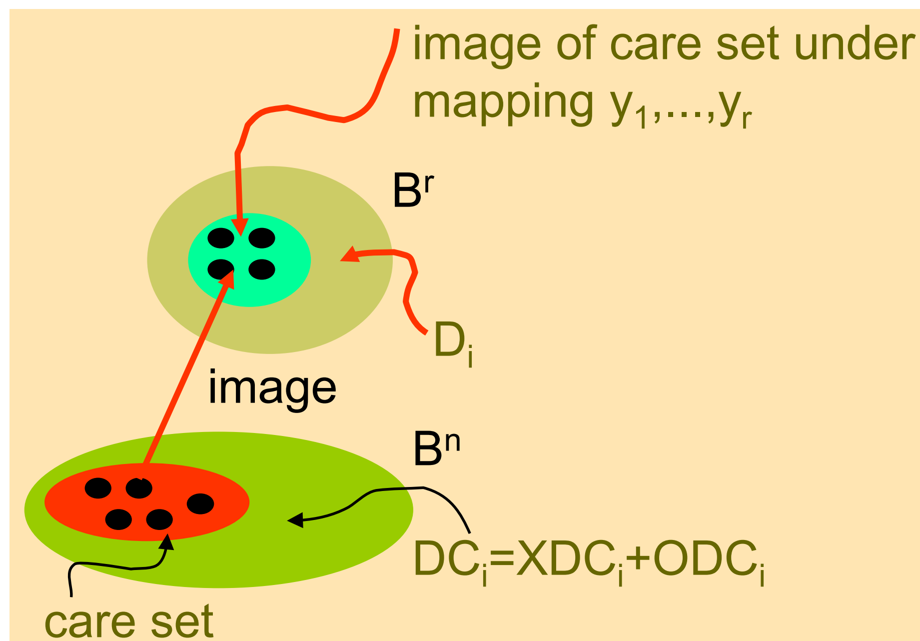
Mapping Don't Cares to Local Space Image Computation

- Local don't cares are the set of minterms in the local space of y_i that cannot be reached under any input combination in the care set of y_i (in terms of the input variables).

- Local don't care set: $D_i = \overline{\text{IMAGE}_{(g_1,g_2,\cdots,g_r)}[\text{care set}]}$

  i.e. those patterns of $(y_1,\ldots,y_r)$ that never appear as images of input cares.

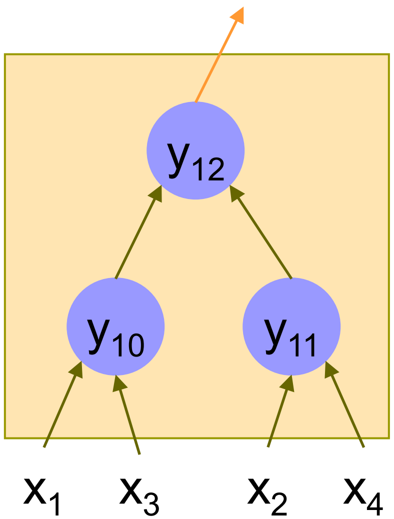


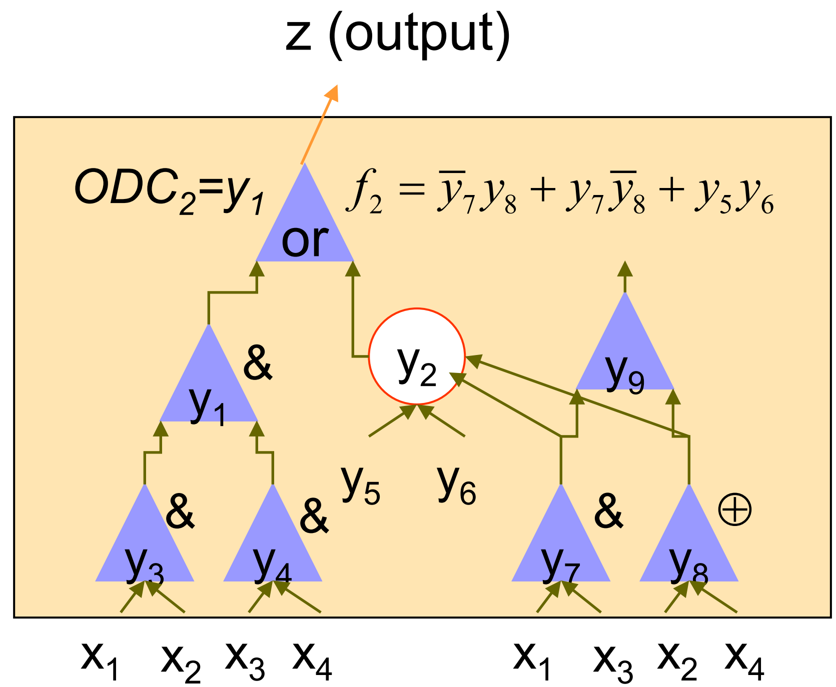


Mapping Don't Cares to Local Space

- Example (cont'd)

XDC $f_{12}=y_{10}y_{11}$





$ODC_2 = y_1$

$ODC_z = y_{12}$

$DC_2 = x_1 x_2 x_3 x_4 + x_1 \bar{x}_2 x_3 \bar{x}_4$

$\overline{DC_2} = \bar{x}_1 + \bar{x}_3 + x_2 \bar{x}_4 + \bar{x}_2 x_4$

$D_2 = y_7 \bar{y}_8$

Note that D_2 is given in this space y_5, y_6, y_7, y_8.

Thus in the space (- - 10) never occurs.

Can check that $\overline{DC_2} D_2 = \emptyset = \overline{DC_2}(x_1 x_3)\overline{(x_2 \bar{x}_4 + \bar{x}_2 x_4)}$

Using $D_2 = y_7 y_8$, f_2 can be simplified to

$f_2 = \bar{y}_7 y_8 + y_5 y_6$

Image Computation

- Two methods:
  1. Transition relation method
     - $f : B^n \rightarrow B^r \Rightarrow F : B^n \times B^r \rightarrow B$

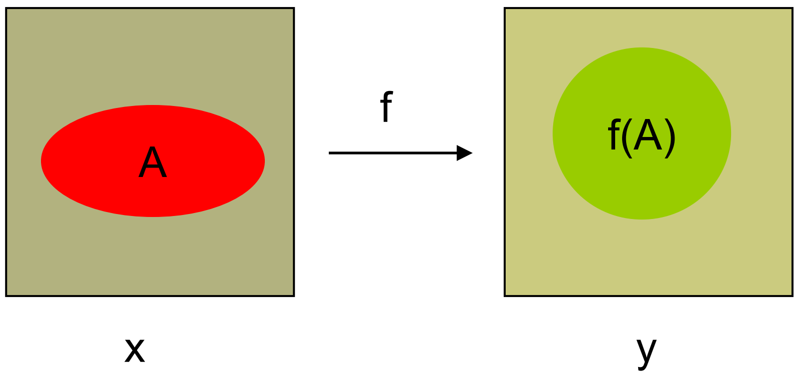
       (F is the characteristic function of f!)

$$
\begin{aligned}
F(x,y) &= \{(x,y) \mid y = f(x)\} \\
&= \prod_{i \le r} (y_i \equiv f_i(x)) \\
&= \prod_{i \le r} (y_i f_i(x) + \overline{y}_i \overline{f}_i(x))
\end{aligned}
$$

  2. Recursive image computation (omitted)

Image Computation
Transition Relation Method

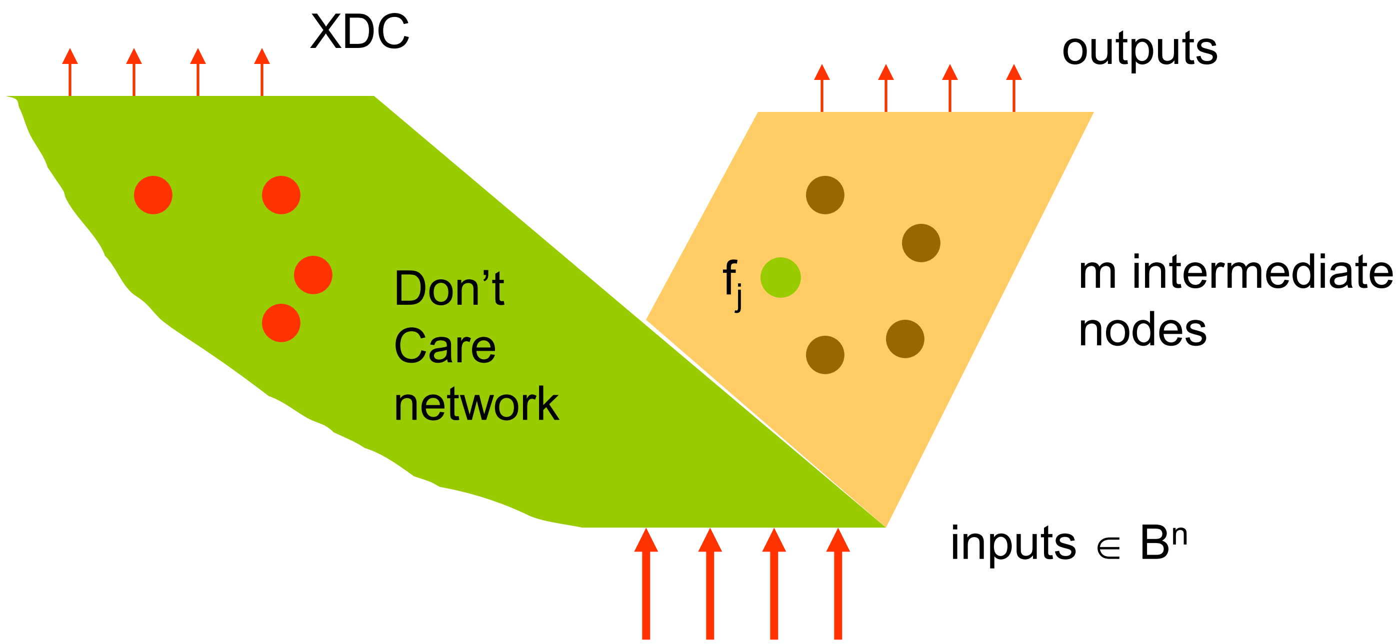
- Image of set A under f: $f(A) = \exists_x(F(x,y) \wedge A(x))$



$$\text{where } \exists_x = \exists_{x_1} \cdots \exists_{x_n} \text{ and } \exists_{x_i} g = g_{x_i} + g_{\bar{x}_i}$$

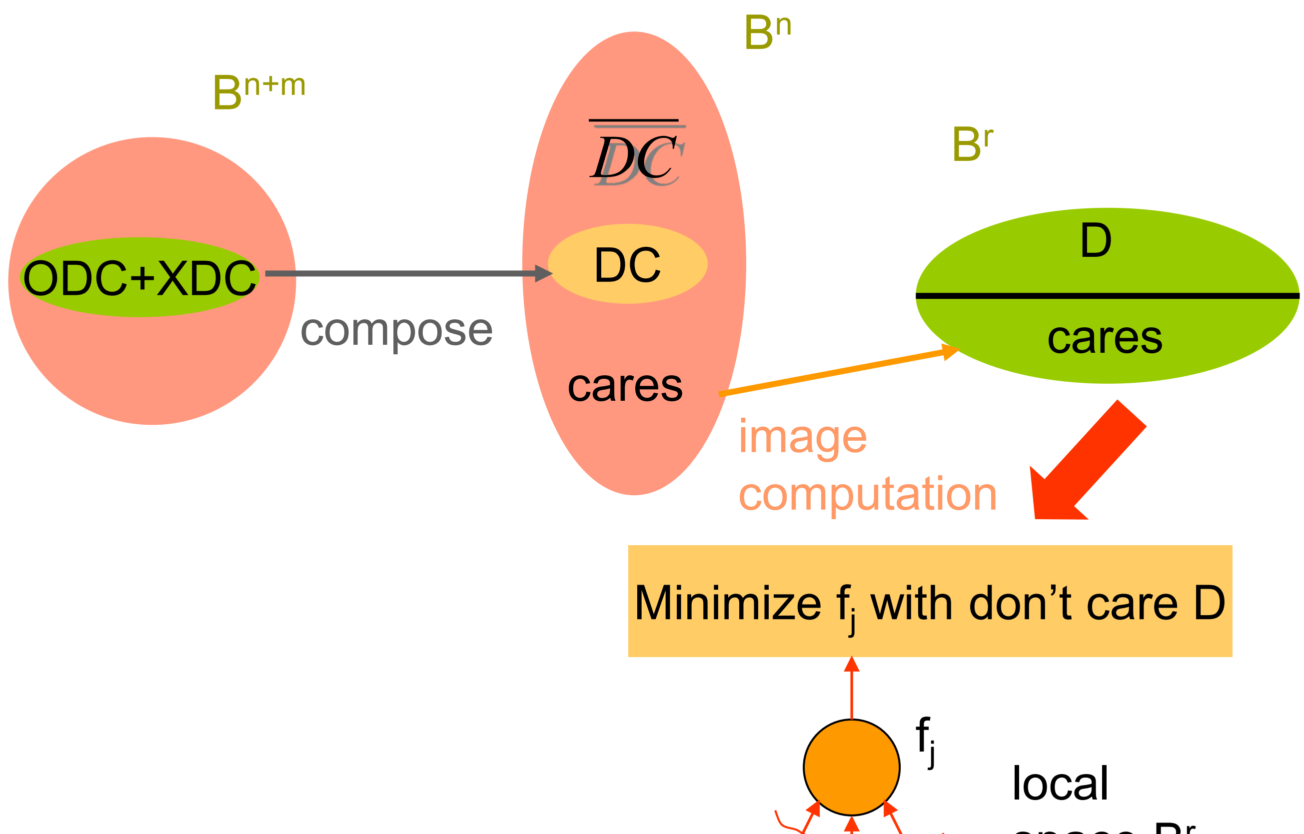
- The existential quantification $\exists_x$ is also called “smoothing”
  Note: The result is a BDD representing the image,
  i.e. f(A) is a BDD with the property that
  BDD(y) = 1 $\Leftrightarrow$ $\exists$x such that f(x) = y and x $\in$ A.

Node Simplification



Express ODC in terms of variables in B^{n+m}

Node Simplification



Question: Where does SDC come into playing?

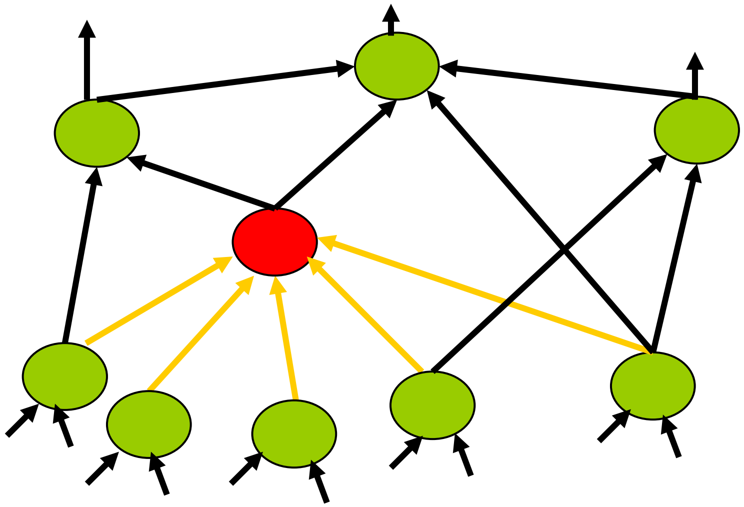
Complete Flexibility

- Complete flexibility (CF) of a node in a combinational network
  - SDC + ODC + localized XDC
  - Used to minimize one node at a time
    - Not considering compatible flexibilities among multiple nodes
    - Different from CODC, where don’t cares at different nodes are compatible and can minimize multiple nodes simultaneously

Complete Flexibility

- **Definition:** A flexibility at a node is a *relation* (between the node's inputs and output) such that *any well-defined sub-relation used at the node* leads to a network that conforms to the external specification

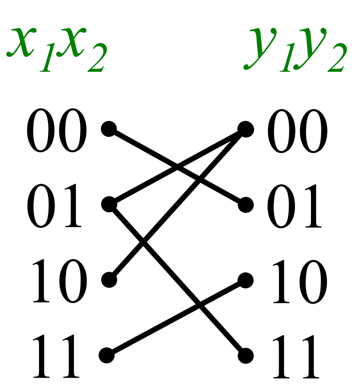
- **Definition:** The complete flexibility (CF) is the *maximum* flexibility possible at a node



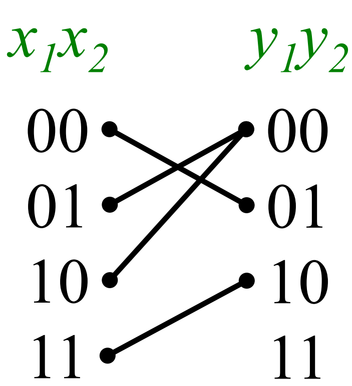
Combinational Logic Network

Relation vs. Function

- Relation *R(X, Y)*
  - Allow one-to-many mappings
    - Can describe non-deterministic behavior
  - More generic than functions



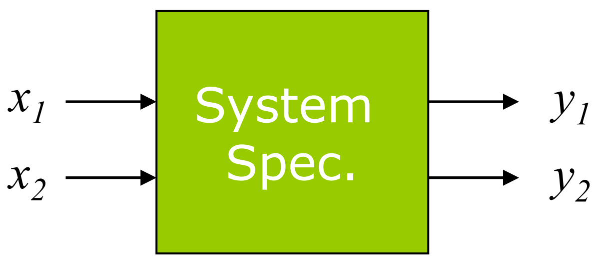
- Function *F(X)*
  - Disallow one-to-many mappings
    - Can only describe deterministic behavior
  - A special case of relation

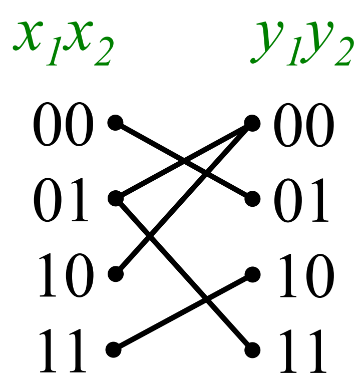


$f_1 = x_1 x_2$
$f_2 = \neg x_1 \neg x_2$

Boolean Relation

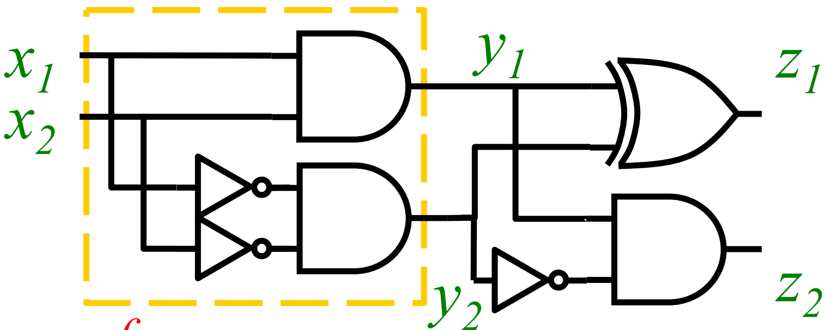
- Applications of Boolean relation
  - In high-level design, Boolean relations can be used to describe (nondeterministic) specifications
  - In gate-level design, Boolean relations can be used to characterize the flexibility of sub-circuits
    - Boolean relations are more powerful than traditional don’t-care representations



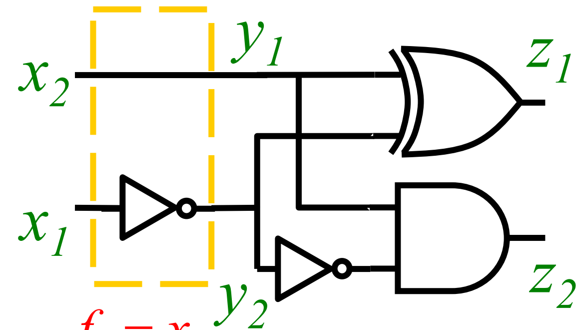


Relation for Circuit Minimization

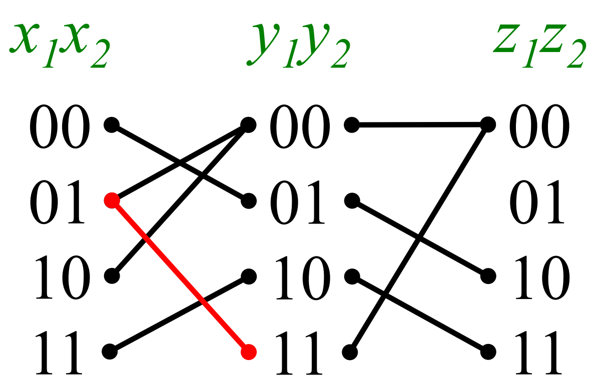
- Example



$f_1 = x_1 x_2$
$f_2 = \neg x_1 \neg x_2$

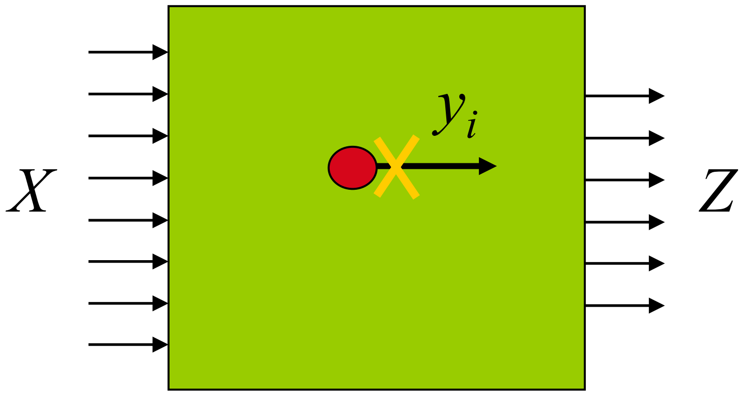


$f_1 = x_2$
$f_2 = \neg x_1$



Complete Flexibility

- Computing complete flexibility



$I(X, y_i, Z)$

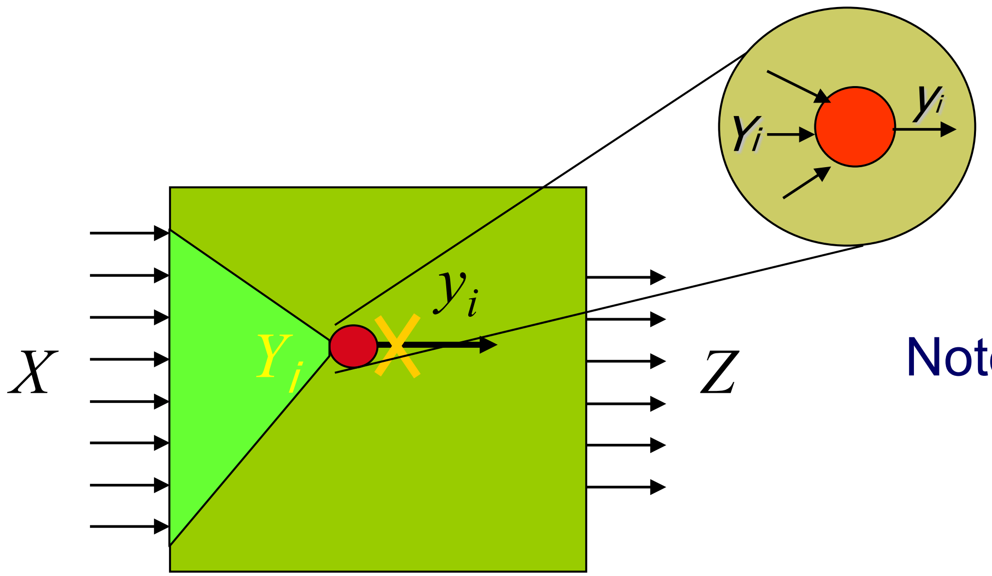
cut the network and treat y_i as a pseudo primary input

$$R(X, y_i) = \forall Z.[I(X, y_i, Z) \Rightarrow S(X, Z)]$$

Note: Specification relation $S(X,Z)$ may contain non-determinism and subsumes XDC. Influence relation $I(X,y_i,Z)$ subsumes ODC.

Complete Flexibility

- Computing complete flexibility

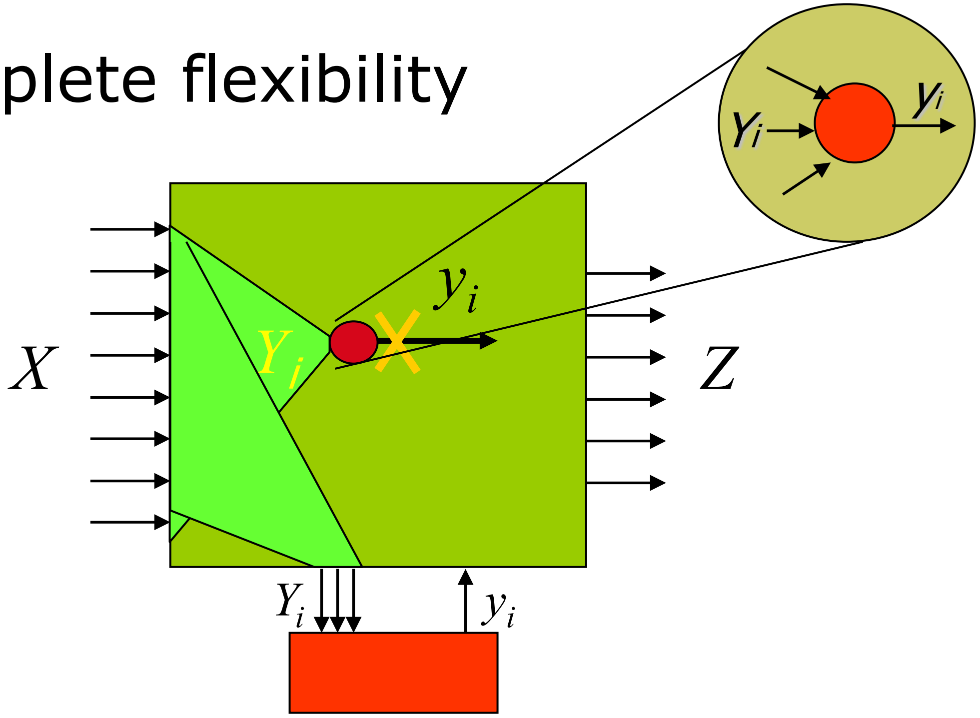


Note: Environment relation $E(X,Y_i)$ subsumes SDC.

$$CF(Y_i, y_i) = \forall X.[E(X,Y_i) \Rightarrow R(X,y_i)]$$
$$= \forall X.[E(X,Y_i) \Rightarrow \forall Z.[I(X,y_i,Z) \Rightarrow S(X,Z)]]$$
$$= \forall X,Z.\neg[E(X,Y_i) \wedge I(X,y_i,Z) \wedge \neg S(X,Z)]$$

Complete Flexibility

- Computing complete flexibility



$$CF(Y_i, y_i) = \forall X.[E(X, Y_i) \Rightarrow \forall Z.[I(X, y_i, Z) \Rightarrow S(X, Z)]]$$

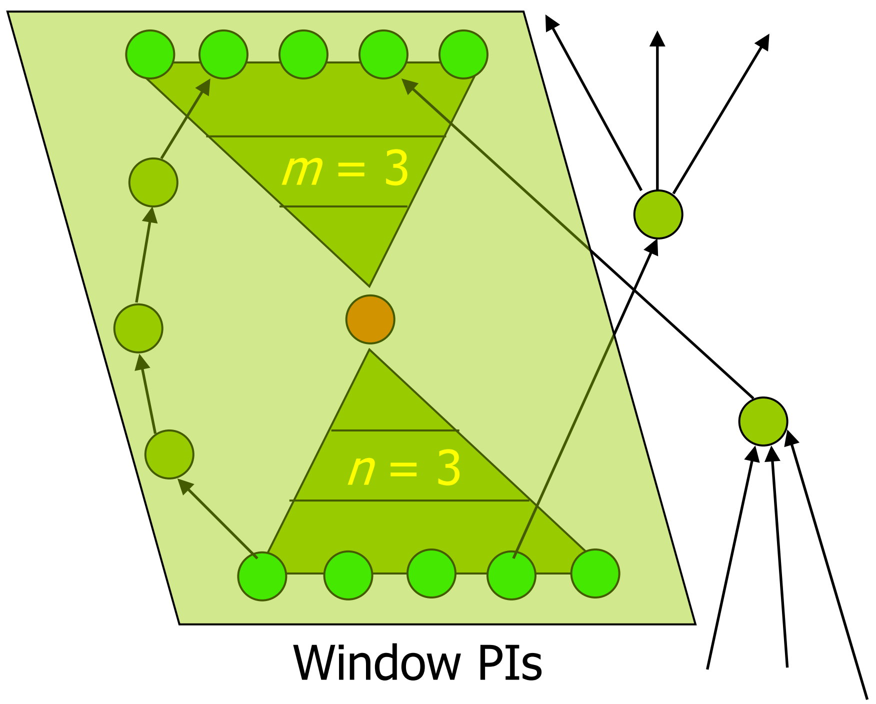
$$= \forall X, Z.[\overline{E(X, Y_i) \cdot I(X, y_i, Z) \cdot \overline{S(X, Z)}}]$$

Note: The same computation works for multiple y_i's

Window and Don't Care Compuation

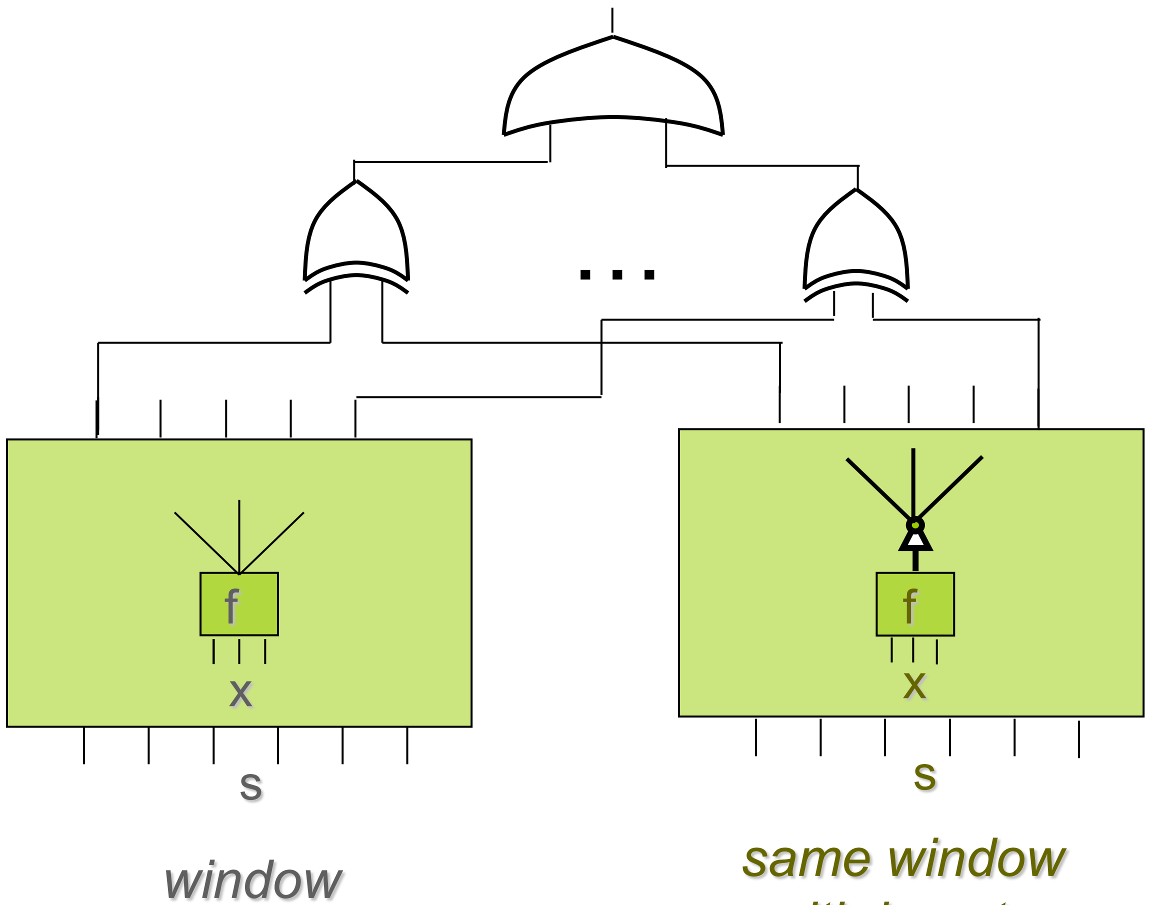
- Definition: A window for a node in the network is the context in which the don't-cares are computed
- A window includes
  - *n* levels of the TFI
  - *m* levels of the TFO
  - all re-convergent paths captured in this scope
- Window with its PIs and POs can be considered as a separate network
- Optimizing a window is more computationally affordable than optimizing an entire network

Boolean network



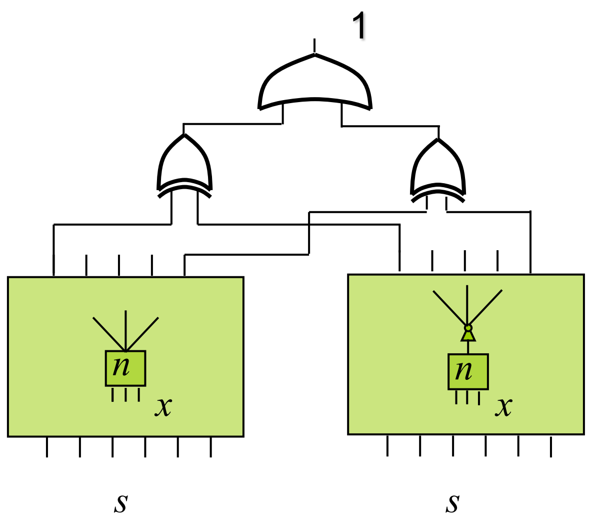
SAT-based Don't Care Computation

"Miter" constructed for the window POs



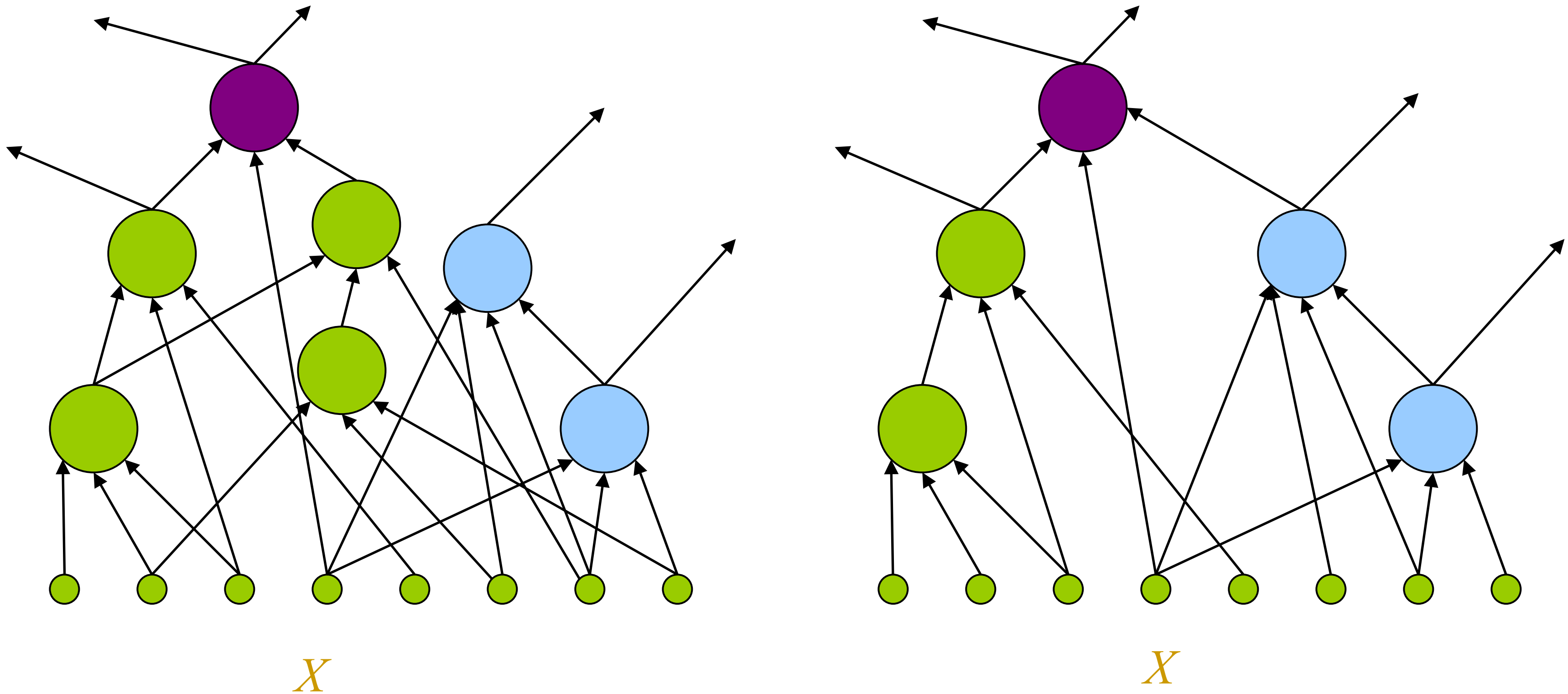
SAT-based Don't Care Computation

- Compute the care set
  - Simulation
    - Simulate the miter using random patterns
    - Collect *x* minterms, for which the output of miter is 1
    - This is a subset of a care set
  - Satisfiability
    - Derive set of network clauses
    - Add the negation of the current care set
    - Assert the output of miter to be 1
    - Enumerate through the SAT assignments
    - Add these assignments to the care set



Resubstitution for Circuit Minimization

- Resubstitution considers a node in a Boolean network and expresses it using a different set of fanins



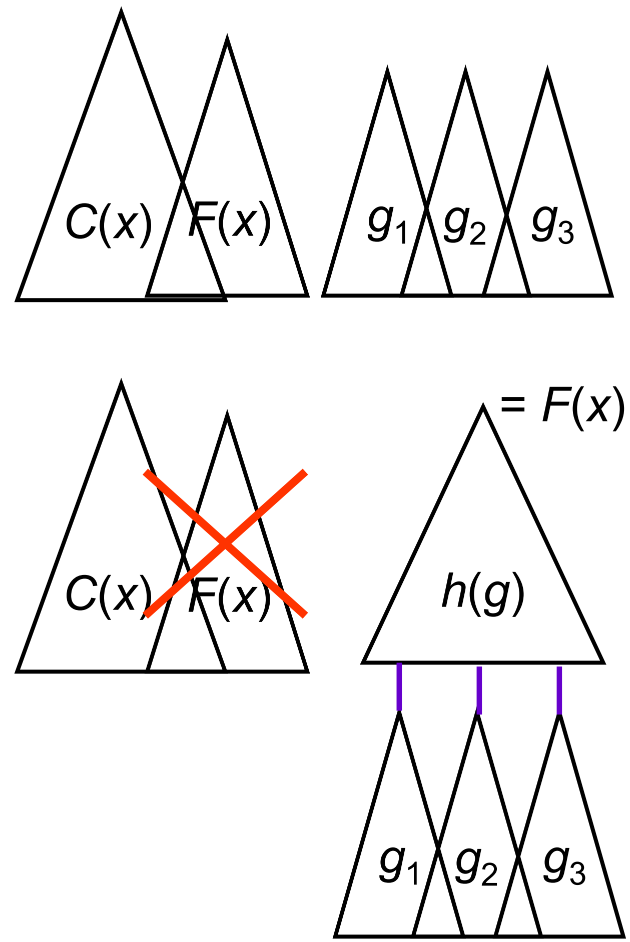
Computation can be enhanced by use of don't cares

Resubstitution with Don't Cares

- Consider all or some nodes in Boolean network
  - Create window
  - Select possible fanin nodes (divisors)
  - For each candidate *subset* of divisors
    - Rule out some subsets using simulation
    - Check resubstitution feasibility using SAT
    - Compute resubstitution function using interpolation
      - A low-cost by-product of completed SAT proofs
  - Update the network if there is an improvement

Resubstitution with Don't Cares

- Given:
  - node function $F(x)$ to be replaced
  - care set $C(x)$ for the node
  - candidate set of divisors $\{g_i(x)\}$ for re-expressing $F(x)$
- Find:
  - A resubstitution function $h(y)$ such that $F(x) = h(g(x))$ on the care set
  - Necessary and sufficient condition: For any minterms a and b, $F(a) \neq F(b)$ implies $g_i(a) \neq g_i(b)$ for some g_i



Resubstitution

- Example

Given:
$F(x) = (x_1 \oplus x_2)(x_2 \vee x_3)$

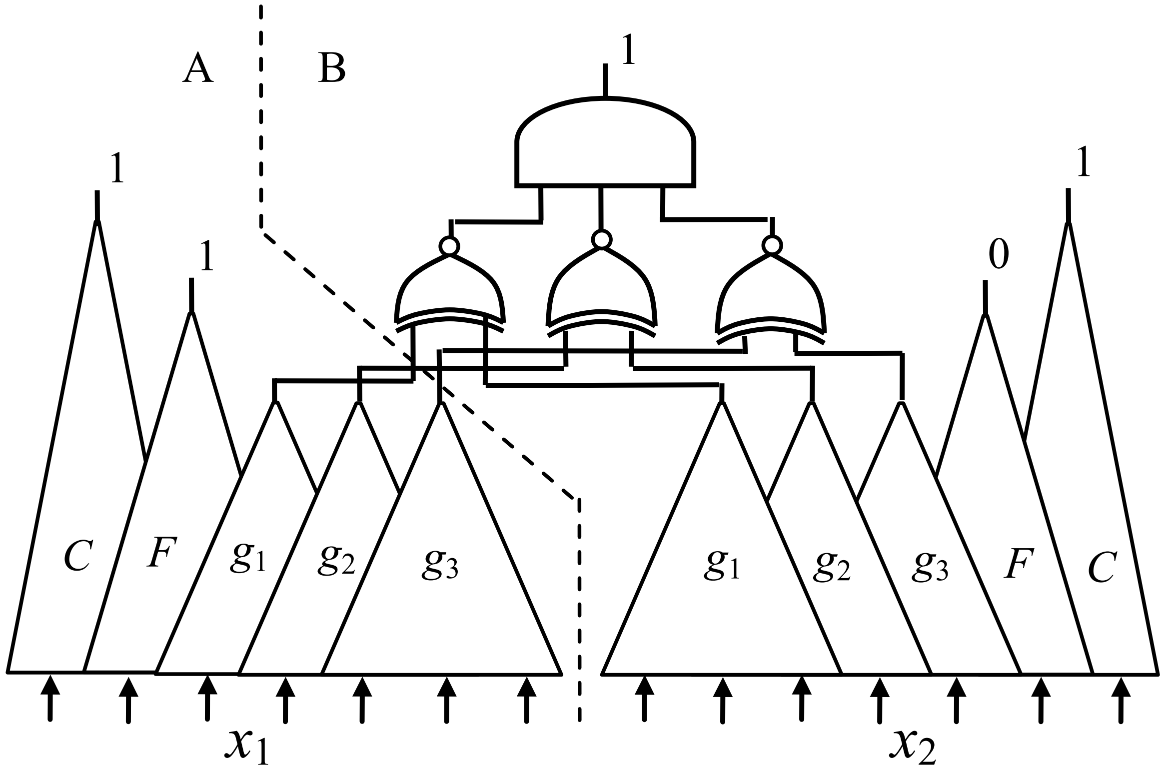
Two candidate sets:
$\{g_1 = x_1'x_2,\ g_2 = x_1\ x_2'x_3\}$,
$\{g_3 = x_1 \vee x_2,\ g_4 = x_2\ x_3\}$

Set $\{g_3, g_4\}$ cannot be used for resubstitution while set $\{g_1, g_2\}$ can.

x	F(x)	$g_1(x)$	$g_2(x)$	$g_3(x)$	$g_4(x)$
000	0	0	0	0	0
001	0	0	0	0	0
010	1	1	0	1	0
011	1	1	0	1	1
100	0	0	0	1	0
101	1	0	1	1	0
110	0	0	0	1	0
111	0	0	0	1	1

SAT-based Resubstitution

Miter for resubstitution check



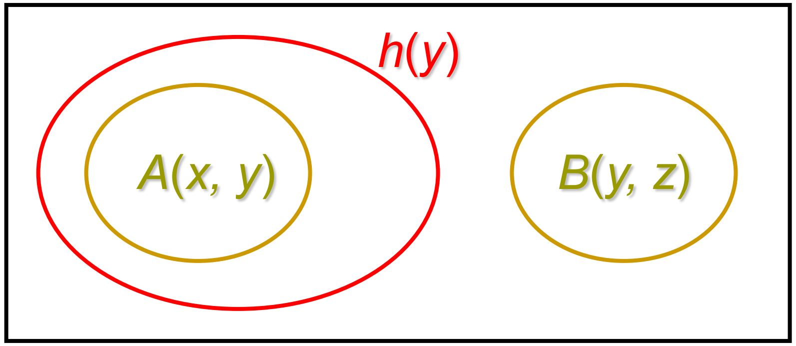
Resubstitution function exists if and only if SAT problem is unsatisfiable

Note: Care set is used to enhance resubstitution check

SAT-based Resubstitution

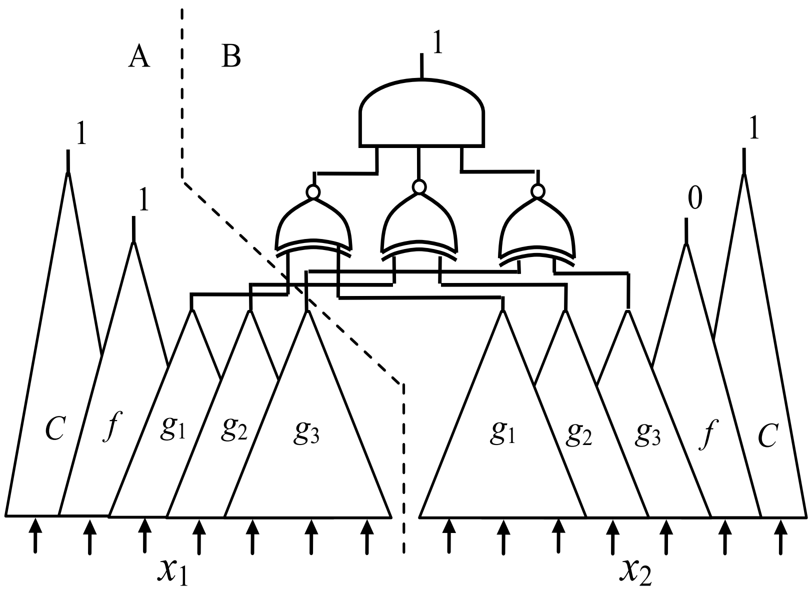
- Computing dependency function h by interpolation
  - Consider two sets of clauses, $A(x, y)$ and $B(y, z)$, such that $A(x, y) \wedge B(y, z) = 0$
  - y are the only variables common to A and B
  - An interpolant of the pair $(A(x, y), B(y, z))$ is a function $h(y)$ *depending only on the common variables* y such that $A(x, y) \Rightarrow h(y) \Rightarrow \neg B(y, z)$

Boolean space (x,y,z)



SAT-based Resubstitution

- Problem: Find function $h(y)$, such that $C(x) \Rightarrow [h(g(x)) \equiv F(x)]$, i.e. $F(x)$ is expressed in terms of $\{g_i\}$
- Solution:
  - Prove the corresponding SAT problem "unsatisfiable"
  - Derive unsatisfiability resolution proof [Goldberg/Novikov, DATE'03]
  - Divide clauses into A clauses and B clauses
  - Derive interpolant from the unsatisfiability proof [McMillan, CAV'03]
  - Use interpolant as the dependency function, $h(g)$
  - Replace $F(x)$ by $h(g)$ if cost function improved



by courtesy of Alan Mishchenko