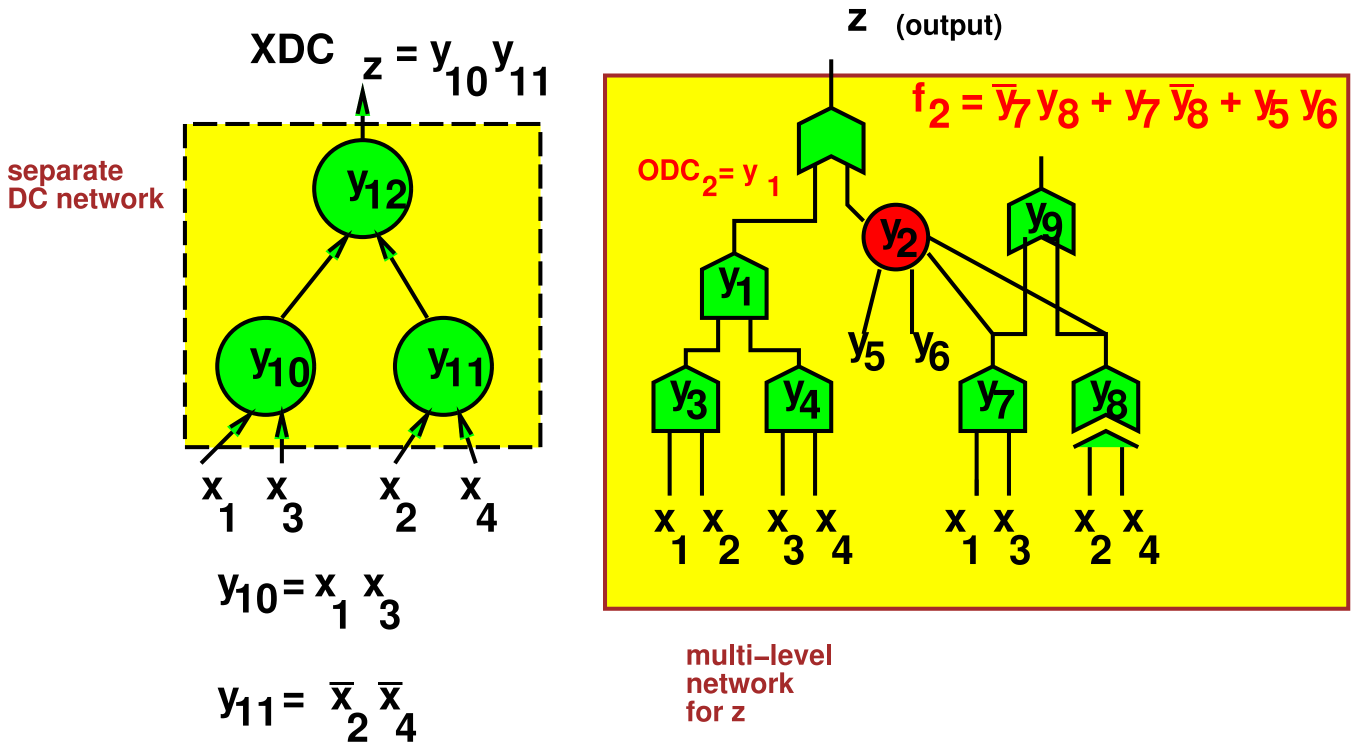
Using Don't Cares - `full_simplify` command

Major command in SIS - uses SDC, ODC, XDC

Key Questions: How do we represent XDC to a network? How do we relate $ODC + XDC$ to a node representation where ODC is some subset of $ODC_j^M(x,y)$?

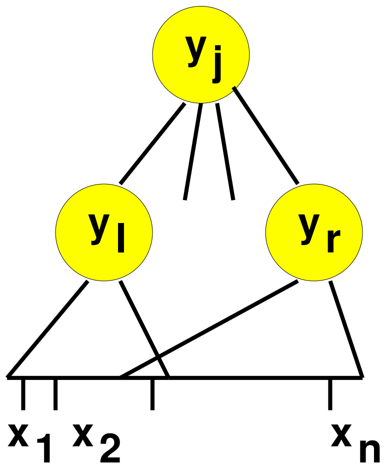


1. XDC is held in a separate network, called the DC network. Described in BLIF by .begin dc ... end dc.

2. XDC and ODC are related **only** through x_1, x_2, x_3, x_4, the primary inputs.

Mapping to Local Space

How can $ODC + XDC$ be used for optimizing the representation of a node?



Definitions:

- The **local space** B^r of node j is the Boolean space of all the fanins of node j (*plus maybe some other variables chosen selectively*).

- A don't care set $D(y^r)$ computed in local space is called a **local don't care set**.

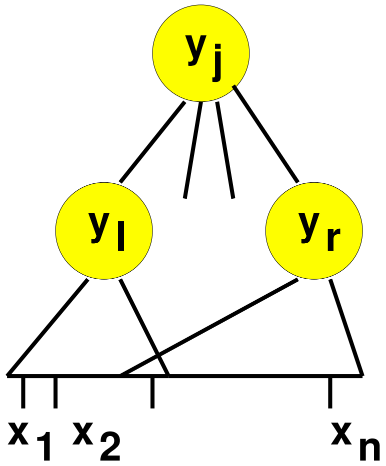
Solution:

- Map $DC(x) = (ODC + XDC)(x)$ to local space of the node to find local don't cares, i.e. we will compute

$$D(y^r) = \overline{\text{IMG}_{g_{fi}}(\overline{DC(x)})}$$

Computing Local Don't Cares

The computation is done in two steps:

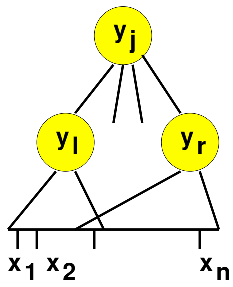


1. Find $DC(x)$ in terms of primary inputs.

2. Find D, the local don't care set, by image computation.

Map to Primary Input (PI) Space

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



$$\begin{aligned} y_j &= f_j(y_k, \ldots, y_l) \\ &= g_j(x_1, \ldots, x_n) \end{aligned}$$

Computation done with Binary Decision Diagrams (BDD's).

- 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 or better, BDD_substitute.

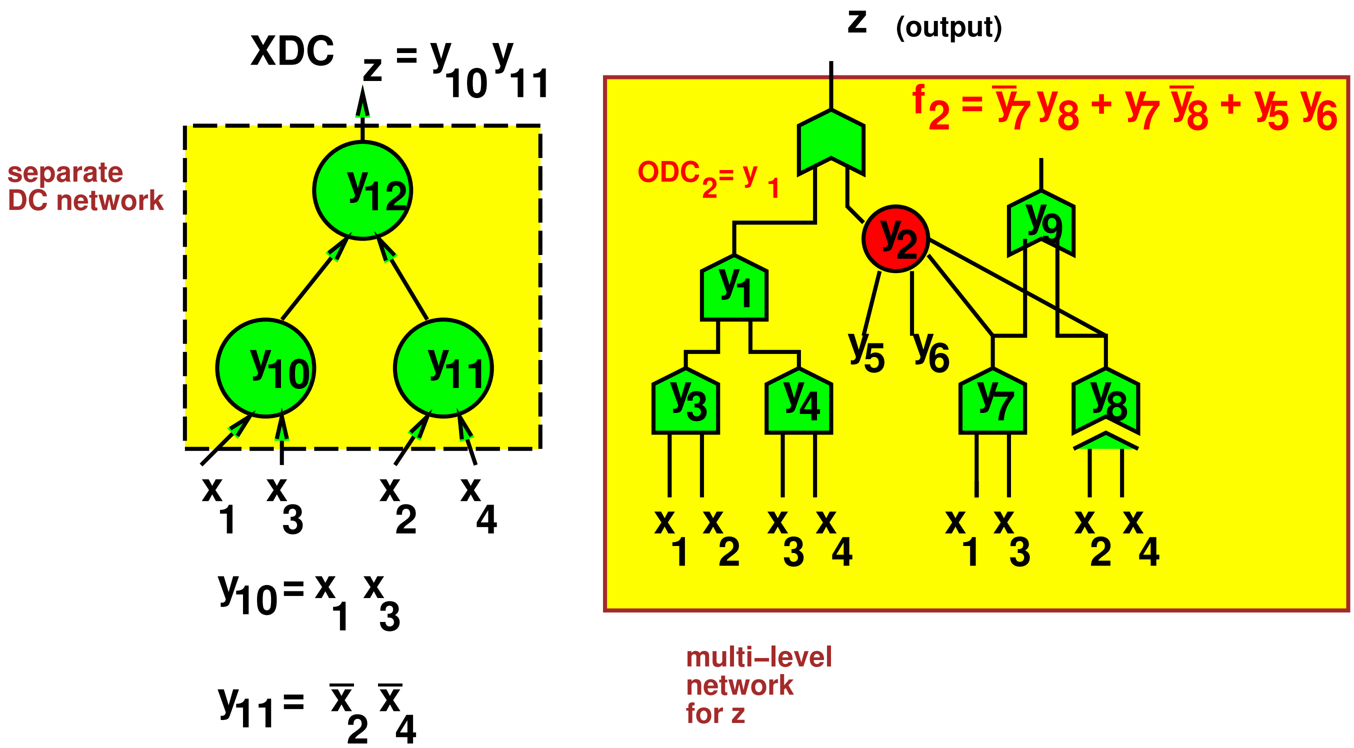
- Replace all the intermediate variables in $(ODC + XDC)$ with their global BDD's
  Since we will be using the method of computing the compatible don't cares by the "local cube method". In general, this will be a set of cubes expressed in (x, y)

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

  Use BDD's to substitute for y in the above (using BDD_substitute)

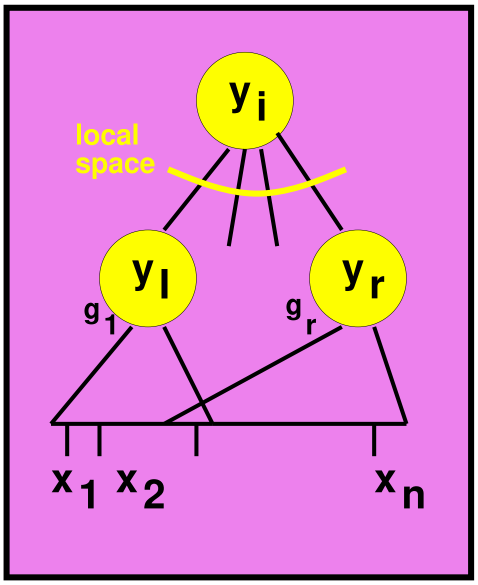
  $$\tilde{h}(x, y) = \tilde{h}(x, g(x)) = h(x)$$

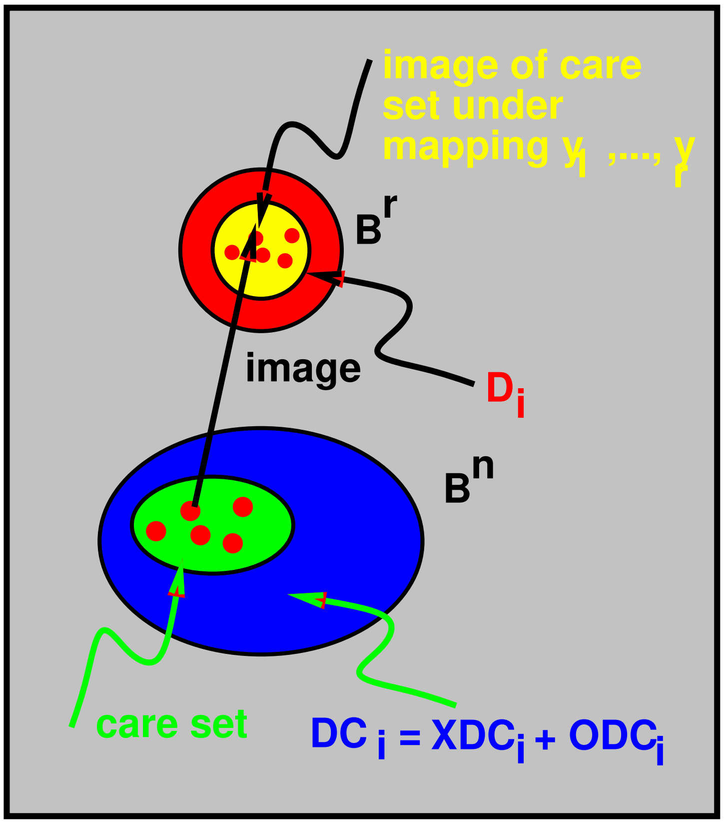
Example



$$
\begin{aligned}
ODC_2 &= y_1 \\
g_1 &= x_1 x_2 x_3 x_4 \\
XDC_z &= y_{12} \\
g_{12} &= x_1 \overline{x}_2 x_3 \overline{x}_4 \\
DC_2 &= x_1 x_2 x_3 x_4 + x_1 \overline{x}_2 x_3 \overline{x}_4 \\
&= ODC_2 + XDC_z
\end{aligned}
$$

Image Computation





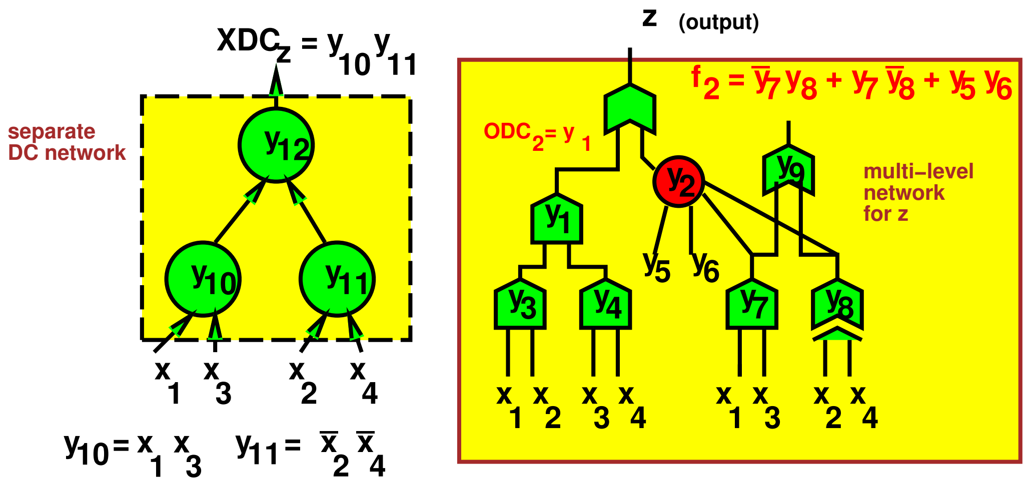
Local don't cares are the set of minterms in the local space of y_i that cannot be reached under any input combination in the care set of y_i.

Local don't Care Set:

$$D_i = \overline{\text{Image}(g_1, g_2, \ldots, g_r)[\text{care set}]}$$

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

Example



$$
\begin{aligned}
ODC_2 &= y_1 \\
ODC_z &= y_{12} \\
DC_2 &= x_1 x_2 x_3 x_4 + x_1 \overline{x}_2 x_3 \overline{x}_4 \\
\overline{DC_2} &= \overline{x}_1 + \overline{x}_3 + x_2 \overline{x}_4 + \overline{x}_2 x_4 \\
D_2 &= y_7 \overline{y}_8
\end{aligned}
$$

Note that D_2 is given in the space y_5, y_6, y_7, y_8. Thus in this space 2210 never occurs.

- Can check that

$$\overline{DC}_2 D_2 = \phi = \overline{DC}_2 (x_1 x_3) \overline{(x_2 \overline{x}_4 + \overline{x}_2 x_4)}$$

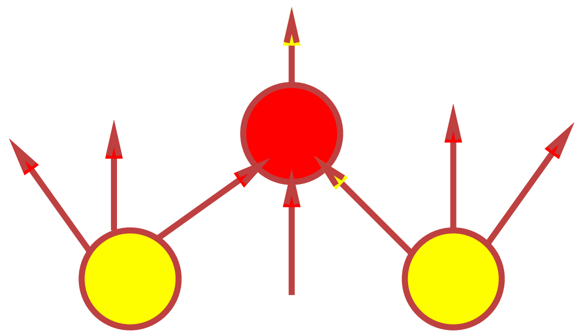
- Using $D_2 = y_7 \overline{y}_8$, f_2 can be simplified to

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

Algorithm

- Visit nodes in **topological reverse order** i.e. from outputs. This gives rise to a total ordering of all nodes. Edge ordering is inherited from node ordering, i.e. $[i,j] < [k,j]$ if $i < k$.

- Compute compatible ODC_i at each node by "flipping" method.
  - compatibility done in SOP form with intermediate y variables.



- BDD's built to get this don't care set in terms of primary inputs (DC_i)

- **Image computation** techniques used to find local don't cares D_i at each node.

$$(XDC_i + ODC_i)(x,y) \rightarrow DC_i(x) \rightarrow D_i(y^r)$$

where ODC_i is a compatible don't care at node i.

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 $y_1, y_2, \ldots$ in that order. The interpretation of the term for 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 can allow it to be a don't care if

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

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

Image Computation - two methods

Transition Relation Method

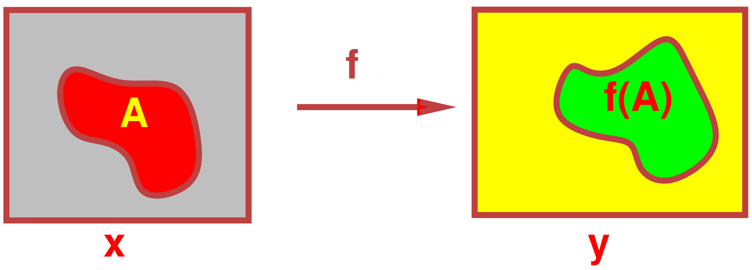
$$f : B^n \to B^r \Longrightarrow F : B^n \times B^r \to B$$

F is the characteristic function of f.

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

Image of set A under f

$$f(A) = \exists_x (F(x,y)A(x))$$



where $\exists_x = \exists_{x_1}, \ldots, \exists_{x_n}$ and $\exists_{x_i} g = g_{x_i} + g_{\overline{x}_i}$.

The existential quantification $\exists_x$ is also called "smoothing" and denoted by $\mathcal{S}_x$.

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 $x \in A$.

Recursive Image Computation

Problem: Given $f : B^n \to B^r$ and $A(x) \subseteq B^n$
Compute: $f|_{x \in A} \equiv \{y \mid y = f(x), x \in A\}$

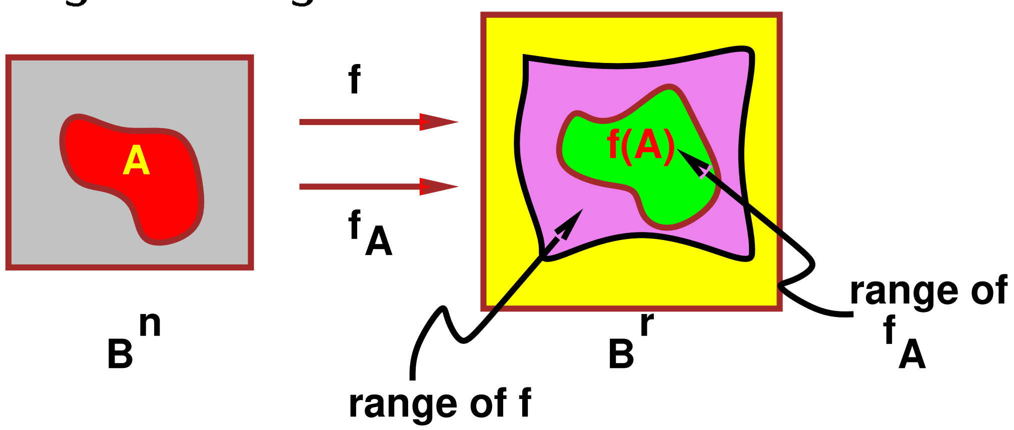
- Step 1: compute **CONSTRAIN**$(f, A(x)) \equiv f_{A(x)}$
  f and A are represented by BDD's. CONSTRAIN is a built-in BDD operation.
  It is related to generalized cofactor with the don't cares used in a particular way to make
  1. *the BDD smaller and*
  2. *an image computation into a range computation.*
- Step 2: compute **range** of $f_{A(x)}$.

$$f_{A(x)} \ : B^n \to B^r$$

Property of **CONSTRAIN**$(f, A) \equiv f_{A(x)}$:

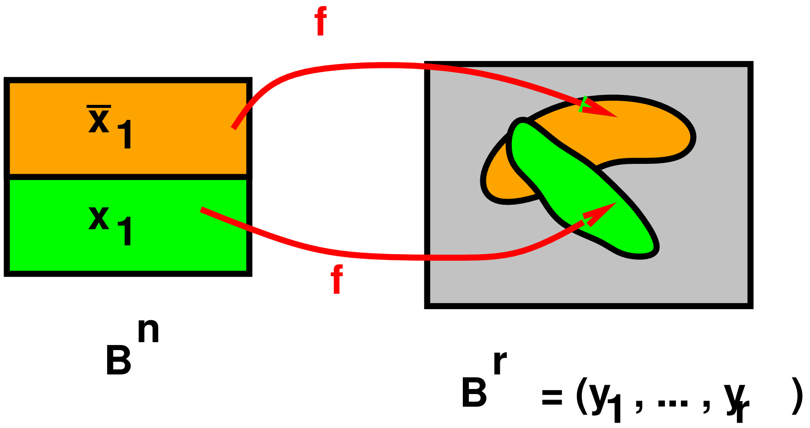
$$(f_{A(x)})|_{x \in B^n} = f|_{x \in A}$$

i.e. **range = image**



Recursive Image Computation

1. $\text{RNG}(f) = \text{RNG}([f_1, f_2, ...f_r]_{x_1}) + \text{RNG}([f_1, f_2, \ldots, f_r]_{\overline{x}_1})$



This is called **input cofactoring**

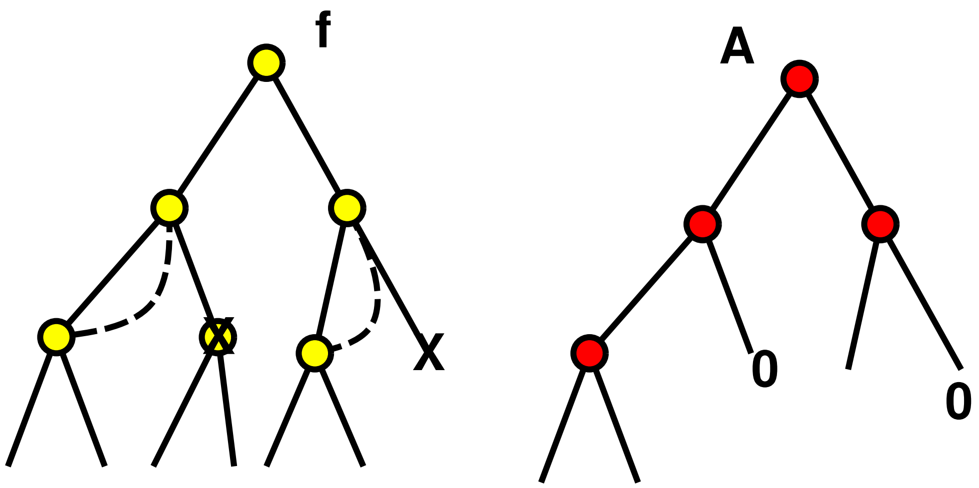
2.
$$
\begin{aligned}
\text{RNG}(f) &= \text{IMG}([f_1, f_2, \ldots, f_r])(x \in f_1^{-1}(1)) \cup \\
&\quad\ \text{IMG}([f_1, f_2, \ldots, f_r])(x \in f_1^{-1}(0)) \\
&= \text{RNG}([1, f_2, ...f_r]_{f_1}) + \text{RNG}([0, f_2, \ldots, f_r]_{\overline{f}_1}) \\
&= y_1 \text{RNG}([f_2, ...f_r]_{f_1}) + \overline{y}_1 \text{RNG}([f_2, \ldots, f_r]_{\overline{f}_1})
\end{aligned}
$$

where here $[\ ]_{f_1}$ refers to the CONSTRAIN operator. (This is called **output cofactoring**). Thus

$$f|_{x \in A}(y^r) \quad = \quad \text{RNG}([f_1, f_2, \ldots, f_r]_{A(x)})$$

Note: This is a recursive call. Could use 1. or 2. Also, note that the input is a set of BDD's and the output can be either a set of cubes or a BDD.

BDD_CONSTRAIN Illustrated

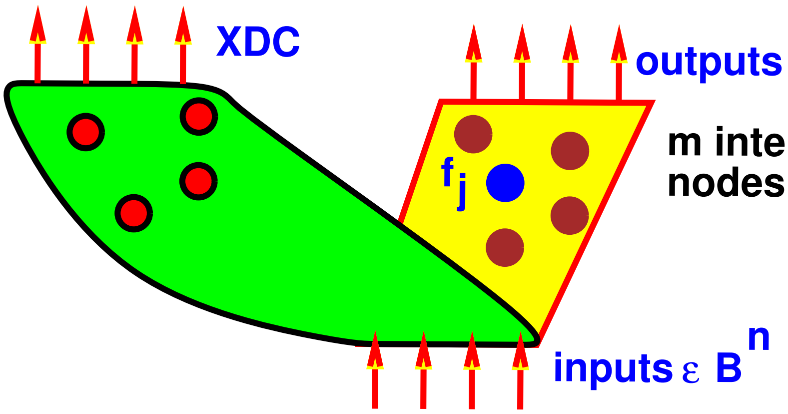


Idea: Map subspace (say cube c in B^n) entirely in DC (i.e. $A(c) \equiv 0$) to nearest non-don't care subspace (i.e. where $A \not\equiv 0$).

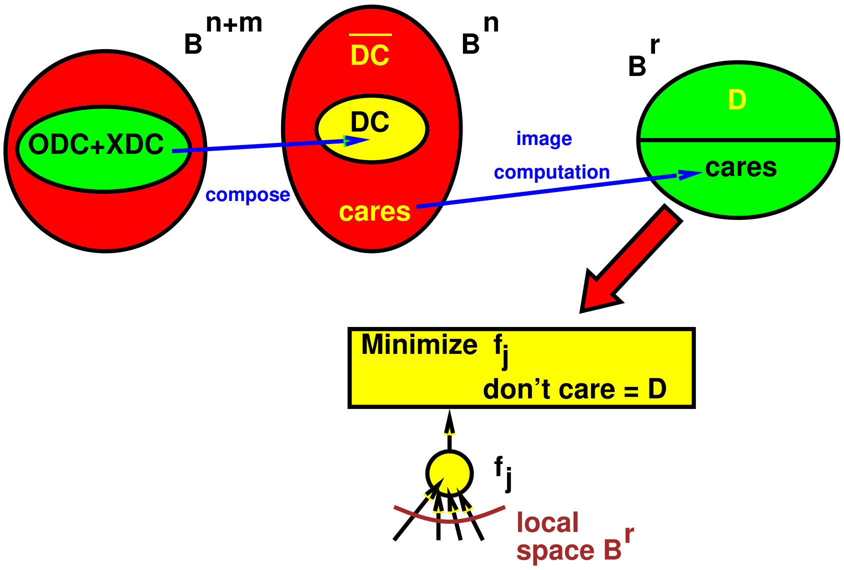
$$f_A(x) = \begin{cases} f(x) & \text{if } A(x) = 1 \\ f(\Pi_A(x)) & \text{otherwise} \end{cases}$$

where $\Pi_A(x)$ is the "nearest" point in B^n such that $A = 1$ there.

full_simplify



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



Question: Where is the SDC coming in and playing a roll?