Two-Level Logic Synthesis
-- Heuristic Method (ESPRESSO)
Exact and Heuristic Minimizers

> Exact Minimizers: guaranteed optimal solution(s)
  = PIs and/or minterms
  = Complexity

> Heuristic Minimizers
  = Quality of the solution
  = Complexity
Local Search

> Approach
1. Find a starting point
2. Search the neighborhood
3. Move to neighbor point(s)
4. Continue until a local optimal is reached

> Components
= Solution space
= Neighborhood structure
= Move strategy
= Action on local optimal
Local Search for Logic Minimization

Goal: Given a Boolean function, find a SOP with minimal cost

Local Search:

- Search space: all SOP forms on given variables
- Solution space: SOP forms equal to the given function
- Neighborhood: two SOP forms are of distance 1 if
  - they have only one product term not in common
  - they have only one literal not in common in this term

How to find neighbors: expansion and reduction

Greedy move strategy: move to a neighbor with less cost

Halt (or try again) at a local minimal
Neighborhood Structure

> Two solutions, $s$ and $t$, are of distance $r$ if there exist a sequence of $(r+1)$ solutions, $s_0, s_1, \ldots, s_r$, s.t. $s_0 = s$, $s_r = t$, and $s_i$ and $s_{i+1}$ are of distance 1.

> (I) $xy + wy'z + w'x'y$  
> (II) $xy' + wy'z + w'x'y$  
> (III) $x'y' + wy'z + w'x'y$

> A local search algorithm usually is based on searching neighbors within a given distance (radius). Larger radius in general leads to more neighbors, better movement, shorter search time before reach a local optimal, but longer run-time overall and no guarantee on the quality of the local optimal.
Expand

> The expansion of an implicant is obtained by deleting one (or more) of its literals.
> A $k$-literal implicant can have $2^k - 1$ expansions.
> Literal vs. minterm
> Check for: Valid Implicant
> Stop at: Prime Implicant

> Goal: expand non-prime implicant to prime with the least literals
Example: Expand

\[ f = x'y'z' + xy'z' + x'yz' + x'y'z \quad \text{with } xyz' \text{ as don't care} \]

\[ = x'y'z' \rightarrow y'z' (xy'z' √) \rightarrow z' (xyz' √, x'yz' √) \rightarrow 1(x) \]

\[ = x'y'z' \rightarrow x'z' (x'yz' √) \rightarrow x' (x'yz x) \]

\[ = x'y'z \rightarrow y'z (xy'z x) \]

> Questions:

= Where to start: one that is unlikely to be covered by others

= How to check validity:
  • compare with off-set
  • compare with on-set + don’t care set

= Which literal to delete?
Reduce

> The **reduction** of an implicant is obtained by adding one or more literals.
> Reduction increases the cost (fanins)
> **Why Reduce?**
>   
>   \[ f = xz' + xy' + x'y + x'z = xy' + x'z + yz' \]  
>   (???, see next slide)
> > Check for: Valid Cover
> > Stop at: Maximally reduced implicant

> Goal: decrease the size of implicants such that expansion may lead to a better solution
Example: Reduce

\[ f = xz' + xy' + x'y + x'z \]

= No implicant can be expanded (I.e., all are PIs)

= Reduce x’y to x’yz’

= Reduce xz’ to xyz’

= Expand x’yz’ to yz’

= \[ f = xy' + x'z + yz' \]

> Reduction is for further expansion, one way to get out of local optimal.
**Irredundant**

> An implicant in a cover is redundant if all the minterms covered by it are contained in other implicants in the cover.

**Irredundant cover vs. minimum cover**

Redundant cover  | Irredundant cover (non-expandable)  | Minimum cover (after reduction + expansion)
--- | --- | ---
redundant PI(s) should be out, but what must be in the cover?
The Espresso Heuristic

> First it applies the Expand and Irredundant operators to optimize the current function specification
> Then it uses the reduce operator to get out of local minimum
> This is iterated till the solution converges
> (This was espresso in a nutshell)