

CS623: CAD FOR VLSI DESIGN
Lecture 34

The objective:

Given a boolean function the problem is to find the minimum prime irredundant cover.

The procedure:

1. Split the function f using shannon expansion, by selecting a variable x_j and minimize recursively f_{x_j} and $f_{\overline{x_j}}$.
2. At the return of the recursive minimization calls, let h_0 stand for the minimized f_{x_j} and h_1 stand for the minimized $f_{\overline{x_j}}$.
3. Realize the minimum prime irredundant cover of $x_j h_0 + \overline{x_j} h_1$.

The next two classes we discuss on

1. The Merging procedure; and,
2. The choice of the splitting variable.

We start with the **Merge** procedure.

The crucial step in the merge procedure is to get minimum prime irredundant cover of f , using the minimum prime irredundant covers of its cofactor with respect to the splitting variable x_j .

To do this:

1. Pick each cube of f got by the computation $x_j h_0 + \overline{x_j} h_1$ and make it as large as possible, compatible with the requirement that it must not contain any vertex that is not in f .
2. Make the cover irredundant by removing all the redundant cubes, i.e. the cubes whose vertices are covered by the remainder of the cover.
3. Of all such irredundant covers, one has to find the minimum.

A cube is enlarged by making a variable with a 0 or a 1 as 2 and the resulting cube will cover the former.

Making the cubes of f as large as possible is straightforward, if h_0 and h_1 , the cubes of the cofactors of f are minimized as prime irredundant covers.

In this case, the only variable that can be changed is x_j . The variable can be changed to 2 in a cube c of $x_j h_0$ where $c = [c_1, c_2, \dots, c_j = a, \dots, c_{n+m}]$, where $a \in \{0, 1\}$, if the cube $\bar{c} = [c_1, c_2, \dots, c_j = \bar{a}, \dots, c_{n+m}]$ is covered by $\bar{x}_j h_1$ and vice-versa.

Keeping in view the number of variables and number of cubes and the fact that computing a irredundant cover is NP-complete, the optimal minimization algorithm is very expensive in terms of time and space consumed.

An approximate Merging is presented. This does not guarantee a minimal prime cover.

Two procedure - MERGE_WITH_IDENTITY (MWI) and MERGE_WITH_CONTAINMENT (MWC) are presented.

MWI checks only for identity between cubes in h_0 and h_1 and factors them out.

Sorting the vectors lexically on their components, the identical cubes h_i of h_0 and h_1 can be filtered out in $O(n \log n)$ time, where $n = |h_0| + |h_1|$.

Then, MWI computes $f = x_j(h_0 - h_i) + \overline{x_j}(h_1 - h_i) + h_i$.

The MWC procedure does the MWI procedure first and in addition checks for each pair of cubes if one **contains** the other.

This procedure takes $O(|h_0||h_1|)$ time.

This can be **heuristically** improved by ordering the cubes in h_0 and h_1 according to the *size* of a cube, using the fact that a smaller cube cannot contain a larger one.

Procedure MERGE_WITH_CONTAINMENT(h_0, h_1)

Begin

$k \leftarrow |h_0|$; $p \leftarrow |h_1|$; $h_2 = \phi$;

for ($i=1, \dots, k$)

for ($l=1, \dots, p$)

Begin

if ($h_0^i \equiv h_1^l$)

Begin

$h_2 \leftarrow h_2 \cup \{h_0^i\}$;

$h_0 \leftarrow h_0 - \{h_0^i\}$;

$h_1 \leftarrow h_1 - \{h_1^l\}$;

End

End

if (CONTAIN == 0) **return** ($h \leftarrow x_j h_0 + \overline{x_j} h_1 + h_2$)

// $ab + a\overline{b} = a$

$k \leftarrow |h_0|$; $p \leftarrow |h_1|$;

for ($i=1, \dots, k$)

for ($l=1, \dots, p$)

Begin

if ($h_0^i \supset h_1^l$)

Begin

$h_2 \leftarrow h_2 \cup \{h_1^l\};$

$h_1 \leftarrow h_1 - \{h_1^l\};$

End

else if ($h_1^l \supset h_0^i$)

Begin

$h_2 \leftarrow h_2 \cup \{h_0^i\};$

$h_0 \leftarrow h_0 - \{h_0^i\};$

End

End

return ($h \leftarrow x_j h_0 + \overline{x_j} h_1 + h_2$)

// $a(x_j + b\overline{x_j}) = a(x_j + b)$

End

Why reduction? The cubes in h_0 and h_1 are independent of the splitting variable x_j and hence would have 2 in the j^{th} component of their input parts.

For the recursive step of the algorithm a **splitting** variable x_j of the function f to be minimized is to be chosen, and Shannon's expansion is applied to get the cofactors.

It is seen that the minimization is efficient if the cofactors turn out to be a **unate** function.

The splitting variable is chosen such that the resulting cofactors are very close to an unate function.

We study the **Unate Functions** and its properties and how to select a **splitting** variable in the next lecture.