Synthesis of 2-level Logic
Heuristic Method

Lecture 8

Two Approaches

• Exact
  – Find all primes
    • Find a complete sum
  – Find a minimum cover (covering problem)

• Heuristic
  – Take an initial cover of cubes
  – Repeat
    • Expand a cube
    • Remove another cube
  – Eliminate consensus terms
The exact method

- Only work for functions with small numbers of inputs
  - For a 16-input function, there can be too many prime implicants to start with
- The constraint matrix can be too big
  - Store it can be a problem
  - Processing it can take a long time
- However, the exact method can still be a good building block in a synthesis framework

Local Search

- Local search is a heuristic method
  - Start with an initial solution
  - Extend the solution to its neighboring solutions
  - Evaluate the solutions to find the best one
  - Repeat until a satisfactory solution is found or run out of time
- Local search does not guarantee finding the optimal solution
  - But it is a practical approach for solving a complex optimization problem
Non-Convex Optimization Problem

- Most problems are non-convex
  - Convex problems can be solved efficiently

Local search for logic minimization

- The initial solution is a set of cubes that cover the function (a cube cover)
- The neighboring solution is obtained by
  - Adding or removing a literal from a cube
    - Ex. Given xyz, we obtain "yz"
- The new cube cover is *feasible* if it is equivalent to the original one
  - How to decide this feasibility quickly?
Example

• Consider a cover (based on 3 variables, x,y,z)
  - \{x'y'z', x'y, yz\}
• Let’s make a move by removing y’ in x’y’z’
  - We obtain \{x'z', x'y, yz\}
  - = \{x'z', yz\}

\[
\begin{array}{c|c}
xyz & f \\
\hline
00 & 1 \\
11 & 1 \\
\end{array}
\]

Example

\[
\begin{array}{c|cc}
xyz & f_1 & f_2 \\
\hline
00 & 10 \\
11 & 11 \\
01 & 01 \\
10 & 01 \\
\end{array}
\]
Example (Cont)

- This move actually increases the cost
  - Because \( f_2 \) now includes 1 more cube

<table>
<thead>
<tr>
<th>( xyz )</th>
<th>( f_1f_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–1</td>
<td>11</td>
</tr>
<tr>
<td>1–0</td>
<td>10</td>
</tr>
<tr>
<td>00–</td>
<td>01</td>
</tr>
<tr>
<td>−00</td>
<td>01</td>
</tr>
<tr>
<td>−11</td>
<td>01</td>
</tr>
</tbody>
</table>

Example (Cont)

- Increasing the cost in a move can lead to decreasing the cost in the final result

<table>
<thead>
<tr>
<th>( xyz )</th>
<th>( f_1f_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–1</td>
<td>11</td>
</tr>
<tr>
<td>1–0</td>
<td>10</td>
</tr>
<tr>
<td>00–</td>
<td>01</td>
</tr>
<tr>
<td>−00</td>
<td>01</td>
</tr>
<tr>
<td>−11</td>
<td>01</td>
</tr>
</tbody>
</table>
Example

\[
\begin{array}{c|cc|cc|c}
\hline
xyz & f_1 & f_2 \\
\hline
1   & 10  & & & & & \\
0   & 10  & & & & & \\
0 & 01  & & & & & \\
-10 & 01  & & & & & \\
\hline
\end{array}
\]

- Add a literal to the second cube
- Again, increasing the cost in a move can lead to reduced-cost result

Minimization loop

\[
F = \text{EXPAND}(F,D);
F = \text{IRREDUNDANT}(F,D);
do \{ 
    \text{cost} = |F|;
    F = \text{REDUCE}(F,D);
    F = \text{EXPAND}(F,D);
    F = \text{IRREDUNDANT}(F,D);
\} \text{while}(|F| < \text{cost});
F = \text{MAKE\_SPARSE}(F,D);
\]

- Increase the overlap of the cubes
- Remove redundant cubes
- Reducing the cubes
Two problems

- So far, we have assumed we could tell valid expansions and reductions
  - This is not trivial for large functions
- Two problems
  - What is the best move out of many?
    - EXPRESSO uses many heuristics to decide on this
  - The new cover is equivalent to the original one?
- Two checks
  - When a cube expands, it covers more min-terms
    - We need to check none of these min-terms belongs to the OFF set
  - When a cube reduces, it removes some min-terms
    - We need to check these removed min-terms are covered by other cubes

Containment check

- Let $F$ be the equation of cubes (product terms) covering of the ON set
- Let $D$ be the equation of cubes (product terms) cover of the DC set
- Let $C_i$ be the cube (as a product term) of $F$ that is altered (expanded or removed)
- Let $\tilde{C}_i$ be the new cube that contains the newly added or removed min-terms
- The expansion or removal is valid iff

$$\tilde{C}_i \subseteq (F-\{C_i\}) \cup D$$
The check is not easy

• Consider $xz \subseteq xy + y'z$

• We **cannot** just scan the list of cubes in $(F-\{C_i\}) \cup D$ and compare it one by one to see if anyone contains $\bar{C_i}$

• Let $f(x_1,x_2,\ldots,x_n)$ be an $n$-input function. Given a cube $C$, let $f_C$ be the cofactor with respect to $C$
  – Theorem: $C \subseteq f \iff f_C = 1$
  – So containment check now becomes **tautology check**!
  – Theorem: $f \equiv 1 \iff fx \equiv 1$ and $fx' \equiv 1$
    • Pick a variable and split

Split and Check

• Let $f$ be a positive unate function in all of its variables
  – Then, $f$ is a tautology iff $f(0,0,\ldots,0) = 1$
  – We can easily generate this to $f$ being a unate function
    • For a unate function, the check is easy

• The split-and-check paradigm in general can still be an exponential algorithm
  – Detect special cases (see page 197)
  – Partition
    • Partition a cover $F = G + H$
    • Where $G$ and $H$ involves different subsets of variables
    • $F$ is a tautology iff $G$ or $H$ is a tautology
Example of partition

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Start from row 1, collect variables \{1,3\}
- From row 3, add variable 4: \{1,3,4\}
- Form a partition as shown

Unate variables

- If F is cover for f that is positive unate in x
  - Then, \( F \equiv 1 \iff F' \equiv 1 \)
- Similarly, If F is cover for f that is negative unate in x
  - Then, \( F \equiv 1 \iff Fx \equiv 1 \)
- With the above two theorems, we can simplify the problem by finding unate variables first
Example

- The cover is positive unate in \( w \) and negative unate in \( x \)
- Set \( w = 0 \) and \( x = 1 \)
- The result is not a tautology

Unate Reduction for Tautology

**Procedure** TAUTOLOGY(\( F, C \))

// \( C \) is a cube returned if \( F \neq 1 \). Then \( C \) contains a minterm \( m \) where \( F(m) = 0 \)

\[ T \leftarrow \text{SPECIALCASES}(F) \]
if \( T \neq -1 \) return \( T \)

\[ F \leftarrow \text{UNATEREDUCTION}(F) \]
if \( F = \emptyset \) print \( C \); return 0;

\[ j \leftarrow \text{BINATESELECT}(F) \]

\[ T \leftarrow \text{TAUTOLOGY}(F_{x_j}, C \cup \{x_j\}) \]
if \( T = 0 \) print \( (C \cup \{x_j\}) \), return 0

\[ T \leftarrow \text{TAUTOLOGY}(F_{x'_j}, C \cup \{x'_j\}) \]
if \( T = 0 \) print \( (C \cup \{x'_j\}) \), return 0

return 1

end
Complementation

- To guide the expansion process, it is often helpful to have the cover of the OFF set
  - This means that we need to get the OFF set cover from \( \{ F \cup D \} \)
- Given a Boolean function \( f \)
  - We have \( f' = x f_x' + x' f_{x'} \)
    - Think about the OBDD
    - How do you do complementation?
  - We can follow a split-and-complement paradigm to construct the complement cover

Example

<table>
<thead>
<tr>
<th>( x = 1 )</th>
<th>( x = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y = 1 )</td>
<td>( y = 1 )</td>
</tr>
<tr>
<td>( z )</td>
<td>( f'_{xy} )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Merge results from \( x = 1 \) and result from \( x = 0 \) and obtain: