CSE 140: Logic Minimization Lecture
What is Logic Minimization?

- Input: A set of minterms corresponding to a function $F$
- Output: A minimal set of prime implicants that corresponds to function $F$
- Example:
  
  Input: $F = abc' + ab'c' + a'bc'$
  Output: $F = ac' + bc'$
Recall the Karnaugh Map

- Adjacent squares are hamming distance 1 away from each other.
- Enables you to visually circle the prime implicants and generate minimal cover.
- Example:

<table>
<thead>
<tr>
<th>wx\yz</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Problem with Karnaugh Maps

- Difficult to trace hamming adjacencies with more than 4 variables.
- Need a way to automate Logic Minimization for a computer to do it.
Logic Minimization Algorithm

- Step 1: Prime Implicant Generation with Tabulation Method
- Conceptually: Expand minterms into larger implicants until you cannot continue expanding.
- Let's start with the first expansion step:
  - Compare minterms pairwise and expand those that are hamming adjacent.
  - For $n$ minterms, this requires $n^2$ comparisons.
  - Can we do better?
Logic Minimization Algorithm

• Observation: a minterm \( m \) that is hamming adjacent to minterm \( n \) will only differ by a single 1.
  • Example: Minterm 001 and 011 are hamming adjacent and can be combined into 0-1.

• Only way for minterms to be hamming adjacent is if one has exactly one more 1 than the other.

• Idea: Partition the minterms into groups based on the number of 1's they contain
Logic Minimization Algorithm

- **Example:**
  - Minterms 0001, 0101, 0110, 1000, 1100, 1101, 1110, 1111
  - Partition into groups Gi, where i is the number of 1's every element in the group contains
    - G1: 0001, 1000
    - G2: 0101, 0110, 1100
    - G3: 1101, 1110
    - G4: 1111
  - Now we only need to combine elements from Gi with ones from Gi+1, reducing the number of comparisons from 64 to 14!
Logic Minimization Algorithm

- Definition: a k-subcube is an implicant with k dashes. For example, 11-0 is a 1-subcube.
- G1: 0001, 1000
  G2: 0101, 0110, 1100
  G3: 1101, 1110
  G4: 1111
- The above comprise the set of 0-subcubes, and we now need to combine them to generate 1-subcubes.
Logic Minimization Algorithm

- 0-subcubes
  G1: \textbf{0001}, 1000
  G2: \textbf{0101}, 0110, 1100
  G3: 1101, 1110
  G4: 1111

- 1-subcubes
  G1: 0-01,
  G2: 
  G3: 
  G4: N/A

- Can combine 0001 and 0101 to get 0-01. 0001 and 0101 are thus covered (denoted with an x)
Logic Minimization Algorithm

- 0-subcubes
  G1: 0001x, 1000
  G2: 0101x, 0110, 1100
  G3: 1101, 1110
  G4: 1111

- 1-subcubes
  G1: 0-01,
  G2:
  G3:
  G4: N/A

- Can't combine 0001 with 0110 because they are not hamming adjacent
Logic Minimization Algorithm

- **0-subcubes**
  G1: 0001\(x\), 1000
  G2: 0101\(x\), 0110, 1100
  G3: 1101, 1110
  G4: 1111

- **1-subcubes**
  G1: 0-01,
  G2:
  G3:
  G4: N/A

- Can't combine 0001 with 1100
Logic Minimization Algorithm

- **0-subcubes**
  - G1: 001x, **1000**
  - G2: **0101**x, 0110, 1100
  - G3: 1101, 1110
  - G4: 1111

- **1-subcubes**
  - G1: 0-01,
  - G2: 
  - G3: 
  - G4: N/A

- Can't combine 1000 with 0101
Logic Minimization Algorithm

- 0-subcubes
  G1: 0001\(_x\), \textbf{1000}
  G2: 0101\(_x\), \textbf{0110}, 1100
  G3: 1101, 1110
  G4: 1111

- 1-subcubes
  G1: 0-01,
  G2:
  G3:
  G4: N/A

- Can't combine 1000 with 0110
Logic Minimization Algorithm

- 0-subcubes
  G1: 0001x, **1000x**
  G2: 0101x, 0110, **1100x**
  G3: 1101, 1110
  G4: 1111

- 1-subcubes
  G1: 0-01, 1-00
  G2:
  G3:
  G4: N/A

- Can combine 1000 with 1100 to get 1-00. We can thus mark them as covered with an x.
Logic Minimization Algorithm

- 0-subcubes
  G1: 0001x, 1000x
  G2: 0101x, 0110x, 1100x
  G3: 1101x, 1110x
  G4: 1111x

- 1-subcubes
  G1: 0-01, 1-00
  G2: -101, -110, 110-, 11-0
  G3: 11-1, 111-
  G4: N/A

- And so on.
Logic Minimization Algorithm

- 1-subcubes
  G1: 0-01, 1-00
  G2: -101, -110, 110-, 11-0
  G3: 11-1, 111-
  G4: N/A

- Now we want to merge our 1-subcubes into 2-subcubes. We can do the same thing, but this time treating the '-' as essentially another symbol.
Logic Minimization Algorithm

• 1-subcubes
  G1: **0-01**, 1-00
  G2: **-101**, -110, 110-, 11-0
  G3: 11-1, 111-
  G4: N/A

• 2-subcubes
  G1:
  G2:
  G3: N/A

• 0-01 and -101 are more than Hamming-1 apart, so can't merge.
Logic Minimization Algorithm

- **1-subcubes**
  G1: **0-01**, 1-00
  G2: -101, **-110**, 110-, 11-0
  G3: 11-1, 111-
  G4: N/A

- **2-subcubes**
  G1: 
  G2: 
  G3: N/A

- 0-01 and -110 are more than Hamming-1 apart, so can't merge.
Logic Minimization Algorithm

- 1-subcubes
  G1: 0-01, 1-00
  G2: -101, -110, 110-, 11-0
  G3: 11-1, 111-
  G4: N/A

- 2-subcubes
  G1:
  G2:
  G3: N/A

- 0-01 and 110- are more than Hamming-1 apart, so can't merge.
Logic Minimization Algorithm

• 1-subcubes
  G1: 0-01, 1-00
  G2: -101, -110, 110-, 11-0
  G3: 11-1, 111-
  G4: N/A

• 2-subcubes
  G1:
  G2:
  G3: N/A

• 0-01 and 11-0 are more than Hamming-1 apart, so can't merge.
Logic Minimization Algorithm

- **1-subcubes**
  - G1: 0-01, **1-00**
  - G2: **-101**, -110, 110-, 11-0
  - G3: 11-1, 111-
  - G4: N/A

- **2-subcubes**
  - G1:
  - G2:
  - G3: N/A

- 1-00 and -101 are more than Hamming-1 apart, so can't merge.
Logic Minimization Algorithm

- 1-subcubes
  - G1: 0-01, 1-00
  - G2: -101, -110, 110-, 11-0
  - G3: 11-1, 111-
  - G4: N/A

- 2-subcubes
  - G1:
  - G2:
  - G3: N/A

- 1-00 and -110 are more than Hamming-1 apart, so can't merge.
Logic Minimization Algorithm

- 1-subcubes
  G1: 0-01, 1-00
  G2: -101, -110, 110-, 11-0
  G3: 11-1, 111-
  G4: N/A

- 2-subcubes
  G1:
  G2:
  G3: N/A

- 1-00 and 110- are more than Hamming-1 apart, so can't merge.
Logic Minimization Algorithm

- 1-subcubes
  G1: 0-01, **1-00**
  G2: -101, -110, 110-, **11-0**
  G3: 11-1, 111-
  G4: N/A

- 2-subcubes
  G1:
  G2:
  G3: N/A

- 1-00 and 11-0 are more than Hamming-1 apart, so can't merge.
Logic Minimization Algorithm

- 1-subcubes
  G1: 0-01, 1-00
  G2: -101, -110, 110-, 11-0
  G3: 11-1, 111-
  G4: N/A

- 2-subcubes
  G1: N/A
  G2: N/A
  G3: N/A

- Can't merge any of G1 into G2!
Logic Minimization Algorithm

• 1-subcubes
  G1: 0-01, 1-00
  G2: -101, -110, 110-, 11-0
  G3: 11-1, 111-
  G4: N/A

• 2-subcubes
  G1: N/A
  G2:
  G3: N/A

• -101 and 11-1 are more than hamming-1 away from each other, so can't merge
Logic Minimization Algorithm

- 1-subcubes
  G1: 0-01, 1-00
  G2: -101, -110, 110-, 11-0
  G3: 11-1, 111-
  G4: N/A

- 2-subcubes
  G1: N/A
  G2: 
  G3: N/A

- -110 and 11-1 are more than hamming-1 away from each other, so can't merge
Logic Minimization Algorithm

- **1-subcubes**
  
  G1: 0-01, 1-00
  G2: -101, -110, 110-, 11-0
  G3: 11-1, 111-
  G4: N/A

- **2-subcubes**
  
  G1: N/A
  G2: N/A
  G3: N/A

- 110- and 11-1 are more than hamming-1 away from each other, so can't merge
Logic Minimization Algorithm

- 1-subcubes
  G1: 0-01, 1-00
  G2: -101, -110, 110-, 11-0\text{x}
  G3: 11-1\text{x}, 111-
  G4: N/A

- 2-subcubes
  G1: N/A
  G2: 11--
  G3: N/A

- 11-0 and 11-1 can combine to form 11--, so they can be merged and marked with an 'x' as covered
Logic Minimization Algorithm

• 1-subcubes
  G1: 0-01, 1-00
  G2: -101, -110, 110-x, 11-0x
  G3: 11-1x, 111-x
  G4: N/A

• 2-subcubes
  G1: N/A
  G2: 11--
  G3: N/A

• And so on.
Logic Minimization Algorithm

- **1-subcubes**
  - G1: 0-01, 1-00
  - G2: -101, -110, 110-x, 11-0x
  - G3: 11-1x, 111-x
  - G4: N/A

- **2-subcubes**
  - G1: N/A
  - G2: 11--
  - G3: N/A

- We then take the remaining uncovered implicants as our prime implicants.
Logic Minimization Algorithm

• Prime implicants:
  0-01 → w'y'z
  1-00 → wy'z'
  -101 → xy'z
  -110 → xyz'
  11-- → wx

• Now need to minimize this.
Logic Minimization Algorithm

- Quine McCluskey Algorithm

- Idea: Create table with minterms and implicants. For example, with our prime implicants that we calculated:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>w'y'z</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wy'z'</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xy'z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xyz'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Logic Minimization Algorithm

- Rule 1: Essential prime implicants must be used, so we can cross out their corresponding rows and columns since they will already be covered.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>w'y'z</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wy'z'</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xy'z</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xyz'</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>wx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Logic Minimization Algorithm

- Rule 2: If one row dominates another row, we can cross out the dominated row because whatever minterms the dominated row contains will already be covered.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Logic Minimization Algorithm

- Rule 3: If one column dominates another column, we can cross out the dominating column because whatever implicant covers the dominating minterm will cover the one it dominates as well.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Logic Minimization Algorithm

• Algorithm
  • Apply the following rules until steady-state
    - Rule 1: Remove essentials to put in our minimal cover and cross out their rows and columns.
    - Rule 2: If a row dominates another row, cross out the dominated row.
    - Rule 3: If a column dominates another column, cross out the dominating column.
Logic Minimization Algorithm

• With our example, we have 4 essentials that result in the entire table going away, so our minimal cover is \( w'y'z + wy'z' + xyz' + wx \).

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w'y'z )</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( wy'z' )</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( xy'z )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( xyz' )</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( wx )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Recall the Karnaugh Map

- Sanity Check
- Minimal cover is: \( w'y'z + wx + xyz' + xy'z' \)
- Same as before!

<table>
<thead>
<tr>
<th>(wx\text{\backslash}yz)</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Logic Minimization Algorithm

• What if we had the following prime implicants?
  P0 = x’y’
  P1 = y’z
  P2 = xz
  P3 = xy
  P4 = yz’
  P5 = x’z’
Logic Minimization Algorithm

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- No row or column dominance! This is called a cyclic core. How do we identify the minimal cover?
Logic Minimization Algorithm

• Use Petrick’s Method!
  • List the implicants in product of sums form
  • What does this mean?
To cover minterm 0, we can use P0 or P5
• To cover minterm 0, we can use P0 or P5

\[ P0 + P5 \]
To cover minterm 1, we can use $P_0$ or $P_1$.

$$(P_0 + P_5)(P_0 + P_1)$$
To cover minterm 2, we can use P4 or P5:

\[(P0 + P5)(P0 + P1)(P4 + P5)\]
Logic Minimization Algorithm

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- We do this for all covered minterms to finally get: 
  \[(P_0+P_5)(P_0+P_1)(P_4+P_5) \ (P_1+P_2)(P_3+P_4) \ (P_2+P_3)\]
Logic Minimization Algorithm

Now that we've formulated it as a product-of-sums, we can convert it to a sum-of-products:

\[(P_0+P_5)(P_0+P_1)(P_4+P_5)(P_1+P_2)(P_3+P_4)(P_2+P_3) =\]
Logic Minimization Algorithm

• Now that we've formulated it as a product-of-sums, we can convert it to a sum-of-products:

\[(P_0+P_5)(P_0+P_1)(P_4+P_5)(P_1+P_2)(P_3+P_4)(P_2+P_3) = (P_0+P_5P_1)(P_4+P_5P_3)(P_2+P_1P_3) = (P_0P_4+P_0P_5P_3+P_1P_4P_5+P_1P_3P_5)(P_2+P_1P_3) = P_0P_2P_4+P_0P_1P_3P_4+P_0P_2P_3P_5+P_0P_1P_3P_5+P_1P_2P_4P_5+P_1P_3P_4P_5+P_1P_2P_3P_5+P_1P_3P_5\]
Logic Minimization Algorithm

• Now that we've formulated it as a product-of-sums, we can convert it to a sum-of-products:

$$(P_0+P_5)(P_0+P_1)(P_4+P_5)(P_1+P_2)(P_3+P_4)
(P_2+P_3) =
(P_0+P_5P_1)(P_4+P_5P_3)(P_2+P_1P_3) =
(P_0P_4+P_0P_5P_3+P_1P_4P_5+P_1P_3P_5)(P_2+P_1P_3) =
$$

$$P_0P_2P_4+P_0P_1P_3P_4+P_0P_2P_3P_5+P_0P_1P_3P_5+
P_1P_2P_4P_5+P_1P_3P_4P_5+P_1P_2P_3P_5+
P_1P_3P_5$$
Logic Minimization Algorithm

• So, there are 2 possible minimum covers:
  P0, P2, P4
  P1, P3, P5

• Recall that:
  P0 = x’y’
  P1 = y’z
  P2 = xz
  P3 = xy
  P4 = yz’
  P5 = x’z’
Logic Minimization Algorithm

- So our minimal covers in sum of products form are:
  \[ x'y' + xz + yz' \]
  \[ y'z + xy + x'z' \]
Logic Minimization Algorithm

• Sanity Check:

<table>
<thead>
<tr>
<th>x\yz</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• Minimal Cover:
  \(x'y' + xz + yz'\)

• Same as what we got using Petrick’s Method!
Logic Minimization Algorithm

- Sanity Check:

<table>
<thead>
<tr>
<th>x\yz</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Minimal Cover:
y’z + xy + x’z’
- Same as what we got using Petrick’s Method!
Logic Minimization Algorithm

- **Overall algorithm**
  - 1. Generate Prime Implicants using Tabulation Method
  - 2. Map Prime Implicants to Quine McCluskey table and perform the pruning based on the row/column dominance and essential heuristics.
  - 3. If cyclic core is reached, apply Petrick's Method.