CS240 Laboratory 2
Digital Logic

• Circuit equivalence
• Boolean Algebra/Universal gates
• Linux, C, Emacs
• Bitbucket, Mercurial
Circuit Equivalence

Two boolean functions with same truth table = equivalent

When there is an equivalent circuit that uses fewer gates, transistors, or chips, it is preferable to use that circuit in the design

Example:
Given: \( F = A'B' + A'B \) \( Q = A' + A'B + A'B' \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A'B'</th>
<th>A'B</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A'</th>
<th>A'B</th>
<th>A' B'</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

F and Q are equivalent because they have the same truth table.
Identities of Boolean Algebra

- Identity law \(1A = A\) \(0 + A = A\)
- Null law \(0A = 0\) \(1 + A = 1\)
- Idempotent law \(AA = A\) \(A + A = A\)
- Inverse law \(AA' = 0\) \(A + A' = 1\)
- Commutative law \(AB = BA\) \(A + B = B + A\)
- Associative law \((AB)C = A(BC)\)
\((A + B) + C = A + (B + C)\)
- Distributive law \(A + BC = (A + B)(A + C)\)
\(A(B + C) = AB + AC\)
- Absorption law \(A(A + B) = A\)
\(A + AB = A\)
- De Morgan's law \((AB)' = A' + B'\)
\((A + B)' = A'B'\)

Example:
\[F = A'B' + A'B\]
\[= A'(B' + B)\text{ distributive}\]
\[= A'(1)\text{ inverse}\]
\[= A'\text{ identity}\]

\[Q = A' + A'B + A'B'\]
\[= A' + A'B'\text{ absorption}\]
\[= A'\text{ absorption}\]
### Universal Gates

Any Boolean function can be constructed with NOT, AND, and OR gates.

NAND and NOR = **universal gates**

**DeMorgan’s Law** shows how to make AND from NOR (and vice-versa)

\[
\text{AND from NOR: } AB = (A' + B')' \\
\text{OR from NAND: } A + B = (A'B')'
\]

To implement a function using only NOR gates:

- apply DeMorgan's Law to each AND in the expression until all ANDs are converted to NORs
- use a NOR gate for any NOT gates, as well.
- remove any redundant gates (NOT NOT, may remove both)

Implementing the circuit using only NAND gates is similar.
Example: \[ Q = (AB)'B' \]

\[ = (A' + B')B' \]

\[ = ((A'+B')' + B)' \quad \text{NOTE: you can use a NOR gate to produce A’ and you can do the same for B’} \]

Simplifying Circuits or Proving Equivalency

General rule to simplify circuits or prove equivalency:

1. Distribute if possible, and if you can’t, apply DeMorgan’s Law so that you can.
2. Apply other identities to remove terms, and repeat step 1.

EXAMPLE: Is \((A’B’)(AB)’ + A’B’\) equivalent to \((AB)’\)?

\[
F = (A’B’)(AB)’ + A’B’ \quad \text{-- can’t distribute} \\
= (A + B’) (A’ + B’) + A’B’ \quad \text{DeMorgan’s} \\
= AA’ + AB’ + A’B + B’B’ + A’B’ \quad \text{distributive} \\
= 0 + AB’ + A’B + B’ + A’B’ \quad \text{inverse and idempotent} \\
= AB’ + A’B + A’B’ \quad \text{identity} \\
= B’ (A + A’) + A’B \quad \text{distributive} \\
= B’(1) + A’B \quad \text{inverse} \\
= B’ + A’B \quad \text{identity} \\
= B’ + (A + B’)’ \quad \text{DeMorgan’s} \\
= (B(A + B’))’ \quad \text{DeMorgan’s} \\
= (AB + BB’)’ \quad \text{distributive} \\
= (AB + 1)’ \quad \text{inverse} \\
= (AB)’ \quad \text{identity} \]

Demo LogicWorks