Logic

Why Study Logic?
- A basis for computer hardware
- A basis for computer programming
- A basis for program optimization
- A basis for specification
- A basis for verification and testing

In a certain sense
Computing is Logic

Is all Logic Computing?
No, but a lot of it can be reduced to computing.

Flavors of Logic
- Proposition Logic
- Predicate Logic
- Temporal Logic
- Modal Logics
- Programming Logics
- Fuzzy Logic

Proposition Logic
- Also known as Switching Logic
- Basic elements are
  - 0 (false)
  - 1 (true)
- proposition variables (take values 0 or 1)
- either
  - functions (functional view)
  - connectives (expression view)
Mostly we use

- the function view
- and occasionally
- the expression view

Proposition Logic Domain

- \{false, true\} (for purists)
or\{0, 1\} (more readable)
or\{⊥, T\} (more symmetric)

Proposition Logic Functions

- and
- or
- not
- implies
- iff (if, and only if)
- others

and

form 1 table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>and(x, y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

results

form 2 table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>and(x, y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

results

rex "table":

- and(0, 0) ⇒ 0;
- and(0, 1) ⇒ 0;
- and(1, 0) ⇒ 0;
- and(1, 1) ⇒ 1;
and

- shorter rex rules (using sequential convention):
  - `and(1, 1) => 1;`
  - `and(x, y) => 0;`

or

- form 1 table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>or(x, y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- form 2 table:

<table>
<thead>
<tr>
<th>or(x, y)</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

or

- rex “table”
  - `or(0, 0) => 0;`
  - `or(0, 1) => 1;`
  - `or(1, 0) => 1;`
  - `or(1, 1) => 1;`

or

- shorter rex rules:
  - `or(0, 0) => 0;`
  - `or(x, y) => 1;`

not

- form 1 table = form 2 table:

<table>
<thead>
<tr>
<th>x</th>
<th>not(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
not

- rex rules:
  - \(\neg(0) \Rightarrow 1\);
  - \(\neg(1) \Rightarrow 0\);

implies

- form 1 table:
  
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>implies(x, y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- form 2 table:

<table>
<thead>
<tr>
<th>implies(x, y)</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- shorter rex rules (sequential):
  - \(\neg(0) \Rightarrow 1\);
  - \(\neg(1) \Rightarrow 0\);
  - \(\text{implies}(1, 0) \Rightarrow 0\);
  - \(\text{implies}(x, y) \Rightarrow 1\);

Concise Summary

(sequential convention applies)

- \(\text{and}(1, 1) \Rightarrow 1\);
- \(\text{and}(x, y) \Rightarrow 0\);
- \(\text{or}(0, 0) \Rightarrow 0\);
- \(\text{or}(x, y) \Rightarrow 1\);
- \(\neg(0) \Rightarrow 1\);
- \(\neg(1) \Rightarrow 0\);
- \(\text{implies}(1, 0) \Rightarrow 0\);
- \(\text{implies}(x, y) \Rightarrow 1\);
Expression Forms

- Use for greater readability of certain equalities
- Similar to ordinary discourse

Example:

\[(a \land b) \lor (c \land \overline{d})\]

More Logical Equivalences

<table>
<thead>
<tr>
<th>Equation</th>
<th>Simplified</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a \land 0)</td>
<td>0</td>
</tr>
<tr>
<td>(a \land 1)</td>
<td>1</td>
</tr>
<tr>
<td>(a \lor 0)</td>
<td>a</td>
</tr>
<tr>
<td>(a \lor 1)</td>
<td>1</td>
</tr>
</tbody>
</table>

What We’ll Use

- To start, we’ll use \(\land \lor \neg \implies\)
- When we discuss circuits, we’ll use \(\prime\)
More Logical Equivalences

- \( \neg(a \land b) = (\neg b \lor \neg a) \)
- \( \neg(a \lor b) = (\neg b \land \neg a) \)
- \( (a \land \neg a) \land b = a \lor b \)
- \( (a \land (\neg a \lor b)) = a \land b \)

DeMorgan’s Laws

Logical Equivalences for Implies

- \( (a \rightarrow b) = (\neg a \lor b) \)
- \( (a \rightarrow b) = (\neg (a \land \neg b)) \)
- \( (0 \rightarrow b) = 1 \)
- \( (1 \rightarrow b) = b \)
- \( (a \rightarrow 0) = \neg a \)
- \( (a \rightarrow 1) = 1 \)

More Logical Equivalences for Implies

- \( (a \rightarrow bc) = (a \rightarrow b) \land (a \rightarrow c) \)
- \( ((a \rightarrow b) \land (b \rightarrow c)) \implies (a \rightarrow c) \)
- \( (a \rightarrow b) = (\neg b \rightarrow \neg a) \)

Checking Relations using the Boole–Shannon Principle

- Relations hold iff they hold for any substitution of 0 and 1 for the variables (uniformly throughout the expression)
- Therefore, a relation holds if, choosing any variable \( V \), it holds for \( V = 0 \) and for \( V = 1 \).
- But substituting 0 or 1 for a variable often yields simplifications that make the relation obvious.

Example

- Verify \( (a \rightarrow b) = (\neg b \rightarrow \neg a) \)
- Choose \( a \) as the variable.
  - Substituting 0 for \( a \):
    - \( (0 \rightarrow b) = (\neg b \rightarrow \neg 0) \)
    - which simplifies to:
      - \( 1 = (\neg b \rightarrow 1) \), a known equivalence
  - Substituting 1 for \( a \):
    - \( (1 \rightarrow b) = (\neg b \rightarrow 1) \)
    - which simplifies to:
      - \( b = (\neg b \rightarrow 0) \)

Boole and Shannon

- Boole
  - Invented “Boolean algebra” (switching theory)
  - (In modern mathematics, “Boolean algebra” is a more general, abstract, system)
- Shannon
  - Wrote thesis on switching theory
  - Invented “Information theory”
  - Maze-solving mouse
  - Wrote first chess-playing program
  - Wrote paper on the mathematics of juggling
<table>
<thead>
<tr>
<th>Boole and Shannon</th>
<th>Tautologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>George Boole (1815-1864)</td>
<td>- An expression that always evaluates to 1 (true) regardless of what value each variable is assigned is called a <strong>tautology</strong>.</td>
</tr>
<tr>
<td>Claude Shannon (1916-2001)</td>
<td>- The property of being a tautology can be checked using:</td>
</tr>
<tr>
<td></td>
<td>- Truth-table construction</td>
</tr>
<tr>
<td></td>
<td>- Boole-Shannon Principle, recursively</td>
</tr>
<tr>
<td></td>
<td>- Example of a tautology checker (applet):</td>
</tr>
</tbody>
</table>