Boolean Logic

CS 350: Computer Organization & Assembler Language Programming

A. Why?

- Logical operations on bits is the lowest level of computation we do.
- Converting truth tables to expressions tells us how to do logical calculations.

B. Outcomes

At the end of today, you should:

- Be able to perform logical operations on individual bits using truth tables and simplifications.

C. Logical Values, Logical Operations/Connectives

- **Boolean logic** — named after George Boole
- The logical constants are true and false (typically written 1 and 0 or *T* and *F*).
- The logical operations/connectives are functions on 1 or 2 logical values.
  - Analogous to +, –, etc. on numbers.
  - Unary: *NOT*; Binary: *AND*, *OR*, *XOR*, *NAND*, *NOR*, *IMPL*, *IFF*; we'll often use symbols: ¬ or an overbar for NOT, ∧ or * or juxtapositioning for *AND*, ∨ or + for *OR*, ⊕ for *XOR*, $\overline{\wedge}$ for *NAND*, $\overline{\vee}$ for *NOR*, → for *IMPL* and ↔ for *IFF*.

The Basic Boolean Operations

- $\neg X = 1$ iff $X = 0$
- $X \wedge Y = 1$ iff $X = Y = 1$
  - $X \wedge Y$ is the **conjunction** of X and Y; its conjuncts are X and Y.
- $X \vee Y = 1$ iff $X = 1$ or $Y = 1$ or both ("inclusive" or)
  - $X \vee Y$ is the **disjunction** of X and Y; its disjuncts are X and Y.
- $X \oplus Y = 1$ iff $X \neq Y$. [I.e., one is 1, the other is 0 ("exclusive" or)]

  - The disjuncts of $X \barwedge Y$ are X and Y.
- $X \barwedge Y = 0$ iff $X = Y = 1$. [I.e., $X \barwedge Y = \neg(X \wedge Y)$]
- $X \barvee Y = 1$ iff $X = Y = 0$. [I.e., $X \barvee Y = \neg(X \vee Y)$]
- $X \rightarrow Y = 0$ iff $X = 1$ and $Y = 0$. [I.e., $X \rightarrow Y = (\neg X) \vee Y$]
  - $\rightarrow$ is the **conditional** operator.
  - If true and false are 1 and 0, then $\rightarrow$ behaves like $\leq$, so $1 \rightarrow 0 = 0$ because $1 > 0$
- $\leftrightarrow$ is the **biconditional**
  - $\leftrightarrow$ behaves like equality: $X \leftrightarrow Y = 1$ iff $X = Y$.
  - $\leftrightarrow$ is the opposite of $\oplus$, so sometimes people use "*XNOR*" to mean "$\leftrightarrow$".
    - ("*XNOR*" because "*NXOR*" isn't pronounceable.)
- **Truth tables** are often used to show the possible results of logical expressions.

X	Y	$X \wedge Y$	$X \vee Y$	$X \oplus Y$	$X \barwedge Y$	$X \barvee Y$	$X \leftrightarrow Y$	$X \rightarrow Y$
0	0	0	0	0	1	1	1	1
0	1	0	1	1	1	0	0	1
1	0	0	1	1	1	0	0	0
1	1	1	1	0	0	0	1	1

X	$\neg X$
0	1
1	0

Reading Boolean Expressions

- Precedence and associativity rules tell us that e.g., 1 – 2 + 3 / 4 is an abbreviation of (1 – 2) + (3 / 4) but not e.g., (1 – (2 * 3)) / 4.
  - The / operator takes precedence over the + and – operators.
  - Since it appears to the left of the + sign, the – is done before the +.
  - In general, we say that two expressions are **syntactically equal** if they both are abbreviations for the same fully-parenthesized expression.
- Boolean operators also have precedences and associativity rules.

Precedence Rules

- From higher/stronger to lower/weaker, we have $\neg$, $\wedge$ and $\bar{\wedge}$, $\vee$ and $\oplus$, $\rightarrow$, and $\leftrightarrow$.
  - **Example 1**: $\neg X \wedge Y \vee Z$ abbreviates $((\neg X) \wedge Y) \vee Z$.
  - **Example 2**: $\neg(X \wedge \neg Y \vee Y)$ abbreviates $\neg((X \wedge (\neg Y)) \vee Y)$.

Associativity Rules

- For $\wedge$, $\vee$, $\bar{\wedge}$, $\bar{\vee}$, we'll use **left associativity**.
  - $X\ op\ Y\ op\ Z = (X\ op\ Y)\ op\ Z$ by default
- For $\rightarrow$ and $\leftrightarrow$, we'll use **right associativity**.
  - $X\ op\ Y\ op\ Z = X\ op\ (Y\ op\ Z)$ by default

Associative and Commutative Operators

- It's a bit confusing, but people use "associative" two ways.
- "Is an operator left or right associative?"
  - Answers questions like "Does $1 - 2 - 3$ mean $(1 - 2) - 3$ or $1 - (2 - 3)$?" [It means $(1 - 2) - 3$]
  - Purely a syntactic property involving implicit vs redundant parentheses.
- "Is an operator associative?"
  - Answers the question "Do $((X\ op\ Y)\ op\ Z)$ and $(X\ op\ (Y\ op\ Z))$ always have the same value?"
  - A semantic property that tells us when we can alter an expression by changing its parenthesization but preserve its value.
    - $\wedge$, $\vee$ are associative. (On integers, $\times$ and $+$ are associative.)
    - $\bar{\wedge}$, $\bar{\vee}$, $\oplus$, $\rightarrow$, $\leftrightarrow$ are not associative. (On integers, $-$ is not associative.)
    - There are n-ary versions of $\bar{\wedge}$, $\bar{\vee}$, and $\leftrightarrow$ that are associative. For example, we could define $NAND(X, Y, Z)$ to be $\neg(X \wedge Y \wedge Z)$.
- **Commutativity**: An operator is commutative if $(X\ op\ Y)$ and $(Y\ op\ X)$ always have the same value.

- Commutativity a semantic property also; it tells us when we can change an expression by swapping an operator's operands but preserve its value.
  - $\wedge$, $\vee$, $\overline{\wedge}$, $\overline{\vee}$, $\oplus$, $\leftrightarrow$ are commutative
  - $\rightarrow$ is not commutative

Minimal and Full Parenthesizations

- An expression is **minimally parenthesized** if it has no redundant parentheses. E.g., $(X \vee Y) \wedge Z$.
- An expression is **fully parenthesized** if it has parentheses around each operator subexpression ($op\ X$) or ($X\ op\ Y$). E.g., $((X \wedge Y) \vee Z)$.
  - We don't parenthesize individual variables or constants/ E.,g., we wouldn't write $((X) \wedge (\neg (Y))$
  - The parentheses around the entire expression aren't critical.

D. Truth tables for larger expressions

- A truth table for an expression with n variables begins with the variables as the first n columns.
- Each row begins with a combination of true and false values for the variables. There are 2^n rows in the table, one for each combination.
- There are a couple of styles for representing the value of an expression

Style 1: Column headers are for whole expressions.

- This is the style you're more likely to be familiar with. An example is:

X	Y	$\neg Y$	$X \wedge \neg Y$	$X \wedge \neg Y \vee Y$	$\neg(X \wedge \neg Y \vee Y)$
0	0	1	0	0	1
0	1	0	0	1	0
1	0	1	1	1	0
1	1	0	0	1	0

Style 2: Columns headers are operators or variables

- The column under an operator gives the value of the subexpression headed by that operator. We use extra parentheses to make clearer which operands an operator has.
- Here's the table for the same expression:

X	*Y*	¬	*((X*	∧	¬ *Y)*	∨	*Y)*
0	0	1	0	0	1	0	0
0	1	0	0	0	0	1	1
1	0	0	1	1	1	1	0
1	1	0	1	0	0	1	1

- The first style involves more writing, but it's easy to see what each column's expression is. The second style is briefer but a bit harder to read and write.

E. Alternate Notations

- There are many families of symbols that people use for the logical connectives:

Operator	*Alternatives*
AND	∧, *, juxtaposition
OR	∨, +
NAND	$\overline{\wedge}$
NOR	$\overline{\vee}$
XOR	⊕
IMPL	→, ⇒
IFF	↔, ⇔
NOT	~, ¬, !, overbar: $\overline{X}$, prime: X'

- **Note**: When adjacent variables have overbars, be sure to put some space between them: $\overline{X}\ \ \overline{Y}$ means $(\neg X)$ *AND* $(\neg Y)$ or $(\neg X)(\neg Y)$. On the other hand, $\overline{X\,Y}$ (with one long overbar), means $\neg(X \wedge Y)$ or $X \,\overline{\wedge}\, Y$ or *X NAND Y*.

F. Tautologies, Contradictions, and Contingencies

- A logical expression is a
  - **Tautology** if it is always true (regardless of the values we assign its variables). I.e., its truth table has 1 in all rows.
  - **Contradiction** if it is always false (0's in all rows) .
  - **Contingency** if it has a mix of rows with 0's and 1s
    - (I.e., it's neither a tautology nor a contradiction.)
- Be sure to distinguish between "*P* is not a tautology" and "¬ *P* is a tautology"
  - "¬ *P* is a tautology" means the truth table column for ¬ *P* is all 1's
    - This happens exactly when is the truth table column for *P* is all 0's, when happens when *P* is a contradiction.
  - If *P* is a not a tautology, then it is a contradiction or a contingency
    - The truth table column for *P* is not all 1's when the column is all 0's or is a mix of 0's and 1's.
- Similarly, if *P* is a not a contradiction, then it is a tautology or contingency.
  - But if ¬ *P* is a contradiction, then *P* is a tautology.
- And if *P* is not a contingency, then P is a contradiction or tautology.
  - But if ¬ *P* is a contingency, then *P* is a contingency

G. Logical Equivalence

- Expressions *P* and *Q* are **logically equivalent** if their truth table columns match.
- You can substitute logically equivalent expressions for each other.
  - (Substitute equals for equals.)
- *P* and *Q* are equivalent exactly when $P \leftrightarrow Q$ is a tautology, and vice versa.

Logical Expressions

CS 350: Computer Organization & Assembler Language Programming

A. Why?

- Logical operations on bits is the lowest level of computation we do.
- Mathematical calculations on bits can be viewed as logical operations on bits.

B. Outcomes

After this activity, you should be able to

- Perform logical operations on individual bits using truth tables and simplifications.

C. Problem

1. What is the minimal parenthesization for $((\neg X) \vee (\neg Y)) \wedge Z$?
2. What is the full parenthesization for $\neg X \wedge Y \vee Z$?
3. Write two truth tables for $\neg(X \wedge Y)$ and $\neg X \vee \neg Y$.
   a. Write the table using style 1 (column headers are expressions).
   b. Write the table using style 2 (column headers are operators or variables).

   (You should find these expressions to be logically equivalent (have exactly the same truth table columns. In the next lecture, we'll see that this is one of DeMorgan's laws.)
4. Repeat Problem 3 again, on $((X \barwedge Y) \barwedge Z)$ and $(X \barwedge (Y \barwedge Z))$. Why are these expressions not logically equivalent?
5. Repeat Problem 3 on $((X \leftrightarrow Y) \leftrightarrow Z)$ and $((X \wedge Y) \wedge Z) \vee ((\overline{X} \wedge \overline{Y}) \wedge \overline{Z})$. Why are these expressions not logically equivalent?

