BITS Pilani
Pilani Campus

Discrete Structures for Computer Science

Dr. N. L. Bhanu Murthy
Computer Science & Information Systems Department

Foundations of Logic

- *Mathematical Logic* is a tool for working with elaborate *compound* statements.

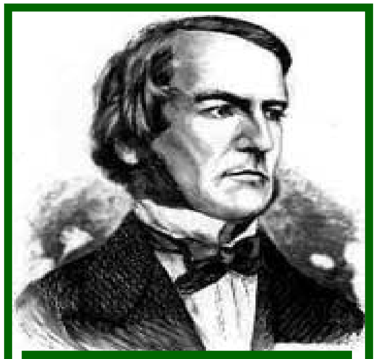
- It includes:
  - ✓ A formal language for expressing them.
  - ✓ A concise notation for writing them.
  - ✓ A methodology for objectively reasoning about their truth or falsity.
  - ✓ It is the foundation for expressing formal proofs in all branches of mathematics.

Foundations of Logic: Overview

- Propositional logic:
  - Basic definitions.
  - Equivalence rules & derivations.
- Predicate logic.
  - Predicates.
  - Quantified predicate expressions.
  - Equivalences & derivations.

Propositional Logic

Propositional Logic is the logic of compound statements built from simpler statements using so-called *Boolean connectives.*



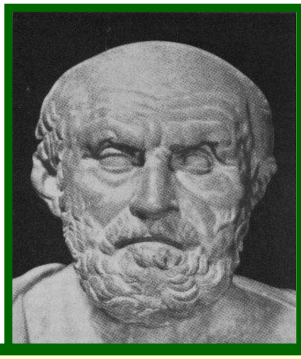
George Boole
(1815-1864)

Some applications in computer science

✓Design of digital electronic circuits.

✓Expressing conditions in programs.

✓Queries to databases & search engines.



Chrysippus of Soli
(ca. 281 B.C. – 205 B.C.)

Definition of a Proposition

Definition: A *proposition* (denoted *p, q, r*, …) is simply:

- ✓ a statement (i.e., a declarative sentence)
  - *with some definite meaning*, (not vague or ambiguous)

- ✓ having a *truth value* that's either *true* (**T**) or *false* (**F**)
  - it is **never** both, neither, or somewhere "in between!"
    - However, you might not *know* the actual truth value,
    - and, the truth value might *depend* on the situation or context.

Examples of Propositions

- “It is raining.” (In a given situation.)
- “Beijing is the capital of China.”
- “1 + 2 = 3”

But, the following are NOT propositions:

- “Who’s there?” (interrogative, question)
- “La la la la la.” (meaningless interjection)
- “Just do it!” (imperative, command)
- “Yeah, I sorta dunno, whatever...” (vague)
- “1 + 2” (expression with a non-true/false value)

Operators / Connectives

- ✓ An *operator* or *connective* combines one or more *operand* expressions into a larger expression. (*E.g.*, "+" in numeric exprs.)
  - ✓ *Unary* operators take 1 operand (*e.g.*, −3);
  - ✓ *binary* operators take 2 operands (*eg* 3 × 4).
- ✓ *Propositional* or *Boolean* operators operate on propositions (or their truth values) instead of on numbers.

Some Popular Boolean Operators

Formal Name	Nickname	Arity	Symbol
Negation operator	NOT	Unary	$\neg$
Conjunction operator	AND	Binary	$\wedge$
Disjunction operator	OR	Binary	$\vee$
Exclusive-OR operator	XOR	Binary	$\oplus$
Implication operator	IMPLIES	Binary	$\rightarrow$
Biconditional operator	IFF	Binary	$\leftrightarrow$

The Negation Operator

- The unary *negation operator* "¬" (*NOT*) transforms a prop. into its logical *negation*.

  E.g. If p = "I have Honda City Car"

  then $\neg p$ = "I do **not** have Honda City Car."

- The *truth table* for NOT:

p	$\neg p$
T	F
F	T

Operand column

Result column

The Conjunction Operator

✓ The binary *conjunction operator* "$\wedge$" (*AND*) combines two propositions to form their logical *conjunction*.

Example

If p="I will have salad for lunch." and
q="I will have steak for dinner.", then
$p \wedge q$="I will have salad for lunch **and** I will have steak for dinner."

AND

Conjunction Truth Table

- Note that a conjunction $p_1 \wedge p_2 \wedge \ldots \wedge p_n$ of n propositions will have 2^n rows in its truth table.

Operand columns

p	q	$p \wedge q$
F	F	F
F	T	F
T	F	F
T	T	T

Remark. ¬ and ∧ operations together are sufficient to express *any* Boolean truth table!

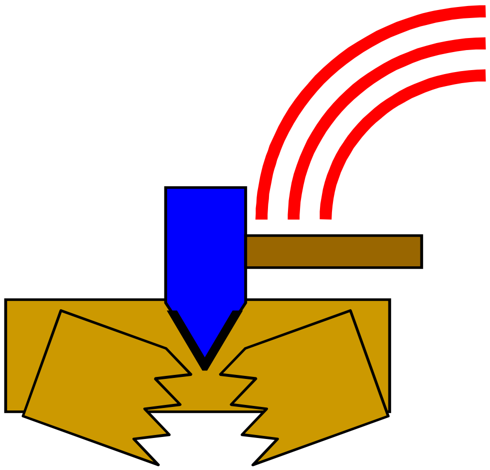
The Disjunction Operator

- The binary *disjunction operator* "$\lor$" (*OR*) combines two propositions to form their logical *disjunction*.

p="My car has a bad engine."

q="My car has a bad carburetor."

$p \lor q$="Either my car has a bad engine, **or** my car has a bad carburetor."



After the downward-pointing "axe" of "$\lor$" splits the wood, you can take 1 piece OR the other, or both.

Meaning is like "and/or" in English.

Disjunction Truth Table

- Note that $p \vee q$ means that p is true, or q is true, **or both** are true!
- So, this operation is also called *inclusive or,* because it **includes** the possibility that both p and q are true.

p	q	$p \vee q$
F	F	F
F	T	**T**
T	F	**T**
T	T	T

Note difference from AND

Nested Propositional Expressions

- Use parentheses to *group sub-expressions*:
  "I just saw my old ***f***riend, and either he's ***g***rown or I've ***s***hrunk." = $f \wedge (g \vee s)$

  - $(f \wedge g) \vee s$ would mean something different

  - $f \wedge g \vee s$ would be ambiguous

- *By convention, "¬" takes precedence over both "∧" and "∨".*

  - $\neg s \wedge f$ means $(\neg s) \wedge f$, **not** $\neg (s \wedge f)$

A Simple Exercise

- Let
  p="It rained last night",
  q="The sprinklers came on last night,"
  r="The lawn was wet this morning."

- Translate each of the following into English:

> - $\neg p$ = "It didn't rain last night."
> - $r \wedge \neg p$ = "The lawn was wet this morning, and it didn't rain last night."
> - $\neg r \vee p \vee q$ = "Either the lawn wasn't wet this morning, or it rained last night, or the sprinklers came on last night."

The Exclusive Or Operator

- The binary *exclusive-or operator* "⊕" (*XOR*) combines two propositions to form their logical "exclusive or" (exjunction?).

- p = "I will earn an A in this course,"
- q = "I will drop this course,"
- $p \oplus q$ = "I will either earn an A in this course, or I will drop it (but not both!)"

Exclusive-Or Truth Table

- Note that $p \oplus q$ means that *p* is true, or *q* is true, but **not both**!

- This operation is called *exclusive or,* because it **excludes** the possibility that both *p* and *q* are true.

p	q	$p \oplus q$
F	F	F
F	T	T
T	F	T
T	T	**F**

Note difference from OR.

Natural Language is Ambiguous

- Note that <u>English</u> “or” can be <u>ambiguous</u> regarding the “both” case!
- “Pat is a singer or Pat is a writer.” -
- “Pat is a man or Pat is a woman.” -

p	q	p "or" q
F	F	F
F	T	T
T	F	T
T	T	?

- Need context to disambiguate the meaning!
- For this class, assume “or” means <u>inclusive</u>.

The Implication Operator

- The *implication* $\underbrace{p}_{\text{antecedent}} \rightarrow \underbrace{q}_{\text{consequent}}$ states that p implies q.

i.e., If p is true, then q is true; but if p is not true, then q could be either true or false.

E.g., let p = "You study hard."
q = "You will get a good grade."
$p \rightarrow q$ = "If you study hard, then you will get a good grade." (else, it could go either way)

Implication Truth Table

- $p \rightarrow q$ is **false** <u>only</u> when p is true but q is **not** true.
- $p \rightarrow q$ does **not** say that p <u>causes</u> q!
- $p \rightarrow q$ does **not** require that p or q **<u>are ever true</u>**!
- *E.g.* "(1=0) $\rightarrow$ pigs can fly" is TRUE!

p	q	$p \rightarrow q$
F	F	T
F	T	T
T	F	**F**
T	T	T

The <u>only</u> False case!

Examples of Implications

- "If Tuesday is a day of the week, then I am a penguin." *True or False?*

- "If 1+1=6, then Anna Hazare is president." *True or False?*

- "If the moon is made of green cheese, then I am richer than Bill Gates." *True or False?*

p	q	$p \rightarrow q$
F	F	T
F	T	T
T	F	**F**
T	T	T

The only False case!

English Phrases Meaning p → q

- "*p* implies *q*"
- "if *p*, then *q*"
- "if *p*, *q*"
- "when *p*, *q*"
- "whenever *p*, *q*"
- "*q* if *p*"
- "*q* when *p*"
- "*q* whenever *p*"
- "*p* only if *q*"
- "*p* is sufficient for *q*"
- "*q* is necessary for *p*"
- "*q* follows from *p*"
- "*q* is implied by *p*"

- We will see some equivalent logic expressions later.

Converse, Inverse, Contrapositive

- Some terminology, for an implication $p \rightarrow q$:
- Its *converse* is: $q \rightarrow p$.
- Its *inverse* is: $\neg p \rightarrow \neg q$.
- Its *contrapositive*: $\neg q \rightarrow \neg p$.
- One of these three has the *same meaning* (same truth table) as $p \rightarrow q$. Can you figure out which?

Contrapositive

How do we know for sure?

- Proving the equivalence of $p \to q$ and its contrapositive using truth tables:

p	q	$\neg q$	$\neg p$	$p \to q$	$\neg q \to \neg p$
F	F	T	T	T	T
F	T	F	T	T	T
T	F	T	F	F	F
T	T	F	F	T	T

The biconditional operator

- The *biconditional* $p \leftrightarrow q$ states that p is true *if and only if (IFF)* q is true.

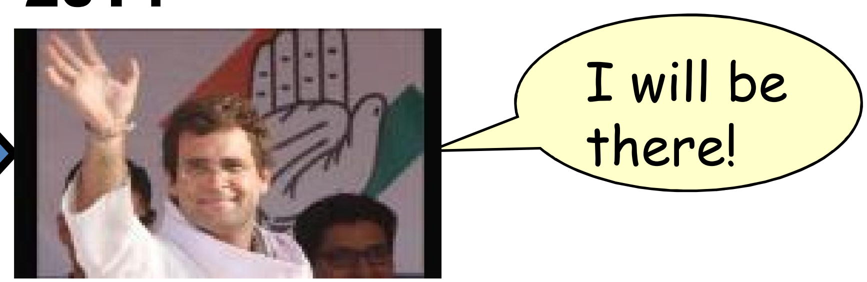
p = “Congress wins 2014 election.”

q = “Rahul will be prime minister for all of us in 2014.”

$p \leftrightarrow q$ = “If, and only if, Congress wins 2014 election, Rahul will be president for all of us in 2014.”

2014

2014



Biconditional Truth Table

- ✓ $p \leftrightarrow q$ means that p and q have the **same** truth value.
- ✓ Remark. This truth table is the exact **opposite** of ⊕'s!
  - ✓ Thus, $p \leftrightarrow q$ means $\neg(p \oplus q)$

p	q	$p \leftrightarrow q$
F	F	T
F	T	F
T	F	F
T	T	T

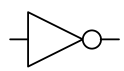
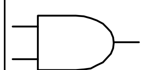
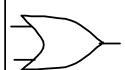
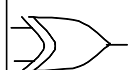
- ✓ $p \leftrightarrow q$ does **not** imply that p and q are true, or that either of them causes the other, or that they have a common cause.

Boolean Operations Summary

- We have seen 1 unary operator (out of the 4 possible) and 5 binary operators (out of the 16 possible).
Their truth tables are below.

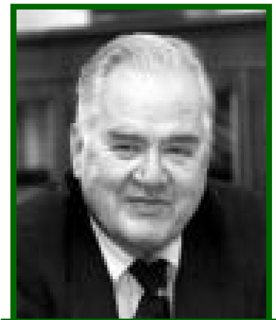
p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \rightarrow q$	$p \leftrightarrow q$
F	F	T	F	F	F	T	T
F	T	T	F	T	T	T	F
T	F	F	F	T	T	F	F
T	T	F	T	T	F	T	T

Some Alternative Notations

Name:	not	and	or	xor	implies	iff
Propositional logic:	$\neg$	$\wedge$	$\vee$	$\oplus$	$\rightarrow$	$\leftrightarrow$
Boolean algebra:	$\overline{p}$	pq	$+$	$\oplus$		
C/C++/Java (wordwise):	!	&&	||	!=		==
C/C++/Java (bitwise):	~	&	|	^		
Logic gates:						

Bits and Bit Operations

- ✓ A *bit* is a binary (base 2) digit: 0 or 1.
- ✓ Bits may be used to represent truth values.
- ✓ By convention:
  0 represents "false";
  1 represents "true".
- ✓ *Boolean algebra* is like ordinary algebra except that variables stand for bits,
  + means "or", and
  multiplication means "and".



John Tukey (1915-2000)

Bit Strings

- ✓ A *Bit string* of *length* n is an ordered sequence (series, tuple) of $n \geq 0$ bits.

- ✓ By convention, bit strings are (sometimes) written left to right:

  - ➢ *e.g.* the “first” bit of the bit string “1001101010” is 1.

  - ➢ Watch out! Another common convention is that the rightmost bit is bit #0, the 2nd-rightmost is bit #1, etc.

- ✓ When a bit string represents a base-2 number, by convention, the first (leftmost) bit is the *most significant* bit. *Ex.* $1101_2 = 8+4+1 = 13$.

Counting in Binary

- ✓ Did you know that you can count to 1023 just using two hands? How?
  - ➢ Count in binary!
  - ➢ Each finger (up/down) represents 1 bit.
- ✓ To increment: Flip the rightmost (low-order) bit.

  If it changes 1→0, then also flip the next bit to the left,

  If that bit changes 1→0, then flip the next one, *etc.*

  0000000000, 0000000001, 0000000010, …
  …, 1111111101, 1111111110, 1111111111

Bitwise Operations

- Boolean operations can be extended to operate on bit strings as well as single bits.

- E.g.:
  01 1011 0110
  <u>11 0001 1101</u>
  Bit-wise OR
  Bit-wise AND
  Bit-wise XOR

End

- You have learned about:
- Propositions: What they are.
- Propositional logic operators'
  - Symbolic notations.
  - English equivalents.
  - Logical meaning.
  - Truth tables.
- Atomic vs. compound propositions.
- Alternative notations.
- Bits and bit-strings.
- Next section: Propositional equivalences.
  - How to prove them.

Thank You!!