# The Logic of Propositions



## Propositional (Boolean) Logic

A proposition is either **True** or **False**

*Example:*

There are 6 regular solids.

**False**

*Non-examples:* Wake up!
Where am I?



## English to Math

Greeks carry Swords or Javelins

$G \rightarrow (S \vee J)$

True even if a Greek carries *both* a Sword and a Javelin



## English to Math

Greeks carry Bronze or Copper swords

$G \rightarrow (B \oplus C)$

Bronze or Copper but not both



## Definition of OR

The value of (P OR Q) is T iff P is T, or Q is T, or *both* are T.

Truth Table for OR

| P | Q | P **OR** Q |
|---|---|---|
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

F iff *both* P,Q are F



## Definition of XOR

The value of (P XOR Q) is T iff *exactly* one of P and Q is T.

Truth Table for XOR

| P | Q | P **XOR** Q |
|---|---|---|
| T | T | F |
| T | F | T |
| F | T | T |
| F | F | F |

## Definition of AND

The value of (P AND Q) is T iff
*both* P and Q are T.

Truth Table for AND

| P | Q | P AND Q |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |



## Definition of NOT

The NOT(P) is T iff P is F.

Truth Table for NOT (P)

| P | NOT(P) |
|---|---|
| T | F |
| F | T |



## Truth Assignments

A truth assignment assigns a value T or F to each propositional variable. Computer scientists call assignment of values to variables an environment. If we know the environment, we can find the value of a propositional formula.



## Evaluation in an Environment

Example: Suppose environment, v, assigns
v(P) = T, v(Q)= T, v(R) = F.

Truth value of
(¬ (P AND Q) ) OR (R XOR (¬ Q))
F T T T F F F F T



## Equivalence

Two propositional formulas are equivalent iff they have the *same truth value* in all environments.



## DeMorgan's Law

$\overline{P \vee Q}$ is equivalent to $\overline{P} \wedge \overline{Q}$

| P | Q | ¬ | (P ∨ Q) |
|---|---|---|---|
| T | T | F | T |
| T | F | F | T |
| F | T | F | T |
| F | F | T | F |

| $\overline{P}$ | ∧ | $\overline{Q}$ |
|---|---|---|
| F | F | F |
| F | F | T |
| T | F | F |
| T | T | T |

## DeMorgan's Law

$\overline{P \vee Q}$ is equivalent to $\overline{P} \wedge \overline{Q}$

| P | Q | ¬(P ∨ Q) | |
|---|---|---|---|
| T | T | F | T |
| T | F | F | T |
| F | T | F | T |
| F | F | T | F |

| $\overline{P}$ | ∧ | $\overline{Q}$ |
|---|---|---|
| F | F | F |
| F | F | T |
| T | F | F |
| T | T | T |

Same final column, so equivalent
-- proof by Truth Table



## Definition of IMPLIES

The value of (P IMPLIES Q) is F iff
P is T and Q is F.

Truth Table for IMPLIES (→)

| P | Q | P→Q |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | T |



## A True Implication

(1=-1) IMPLIES (I am Pope)
We reasoned *correctly* to
reach the false conclusion



## A True Implication

(1=-1) IMPLIES (I am Pope)
We reasoned *correctly* to
reach the false conclusion



## A True Implication

(1=-1) IMPLIES (I am Pope)
We reasoned *correctly* to
reach the false conclusion
from the false hypothesis.



## A True Implication

(1=-1) IMPLIES (I am Pope)
We reasoned *correctly* to
reach the false conclusion
from the false hypothesis.

## A True Implication

(1=-1) IMPLIES (I am Pope)
The *whole implication* is true, even though both conclusion & hypothesis are false.



## Satisfiability & Validity

A formula is satisfiable iff it is true in some environment.

A formula is valid iff it is true in all environments.



## Verifying Valid, Satisfiable

Truth table size doubles with each additional variable --exponential growth. Makes truth tables impossible when there are hundreds of variables. (In current digital circuits, there are millions of variables.)



## Efficient Test for Satisfiability?

The P=NP? question is equivalent to asking if there is an efficient (polynomial rather than exponential time) procedure to check satisfiability.



## Java Logical Expression

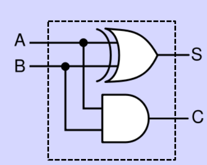
```
                OR          AND
if ((x>0) || (x <= 0 && y>100))
                    :
                (more code)
```



## Digital Logic

s ::= A XOR B

c ::= A AND B



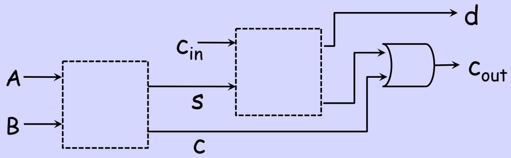

half adder

from http://en.wikipedia.org/wiki/Adder_(electronics)

## Digital Logic

$d ::= c_{in}$ XOR $s$

$c_{out} ::= (c_{in}$ AND $s)$ OR $c$




full adder



## Team Problems

# Problems 1—4