Propositional Logic:
Part I - Semantics

Outline

- What is propositional logic?
- Logical connectives
- Semantics of propositional logic
- Tautologies & Logical equivalence Applications:
  1. Building the world with NAND
  2. Normal Forms & minimizing gate delays
- Logical implication, Valid arguments & Semantic entailment $\models$

A Bit of Notation

Consider negation on the real numbers $\mathbb{R}$:

$$f(x) = -x$$

Then $f : \mathbb{R} \rightarrow \mathbb{R}$ is the *signature* of f meaning f takes a real argument and produce a real.

Here $-$ is a *unary prefix* operator meaning it takes one argument, the number immediately following $-$ (e.g., $-(5) = -5$). So really

$$- : \mathbb{R} \rightarrow \mathbb{R}$$

Similarly $+ : \mathbb{R}^2 \rightarrow \mathbb{R}$

$+$ is a *binary* operator on $\mathbb{R}$ so we could treat it as a prefix operator and write +(3,5)=8.

But this is tedious so we use *infix* notation and write $3 + 5 = 8$.

What is Propositional Logic?

Def: A *proposition* is a statement that is either true or false.

E.g. p:"The prof looks tired."
q:"We're hungry and not able to eat."

Propositional logic is a formal mathematical system for reasoning about such statements.

The first statement p is an *atomic proposition.* It cannot be further subdivided.

The 2nd statement q is a *compound proposition* that's truth depends upon the value of the two atomic propositions:

1. h:"We are hungry." and 2. e:"We are able eat."

The *logical connectives* "and" and "not" determine how the atomic proposition affect q.

Restating q in the formal language of propositional logic:

$$q : h \wedge \neg e$$

Logical Connectives

Let T and F represent *true* and *false* respectively.

Define $\mathcal{V} := \{T, F\}$, the set of possible truth values for a proposition. In the following let p, q be propositional variables.

Negation: $\neg$ (NOT)

$\neg : \mathcal{V} \to \mathcal{V}$

p	$\neg p$
F	T
T	F

A *truth table* is tabular representation of the truth values of a proposition under all possible assignments. The above is the table for $\neg p$. Clearly it defines a function.

Truth tables define the meaning or interpretation propositions. We call this the *semantics* of the propositional logic.

Conjunction: $\wedge$ **(AND)**

$\wedge : \mathcal{V}^2 \rightarrow \mathcal{V}$

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

Other English equivalents: "p but q" - "The students are interested but look bored."

Disjunction: $\vee$ **(OR)**

$\vee : \mathcal{V}^2 \rightarrow \mathcal{V}$

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

Note: This is a "non-exclusive OR". Why?

Conditional: $\rightarrow$ (IMPLIES)

$\rightarrow: \mathcal{V}^2 \rightarrow \mathcal{V}$

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

Other English equivalents: “If p then q”, “p only if q”, “q if p”, “p is sufficient for q”, “q is necessary for p”.

Biconditional: $\leftrightarrow$ (IFF)

$\leftrightarrow$: $\mathcal{V}^2 \rightarrow \mathcal{V}$

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

Other English equivalents: "p if and only if q", "p is equivalent to q", "p is necessary and sufficient for q"

Review: Precedence of Arithmetic Operators

We write: $-5 \cdot 2 + 10/5 - 8$ and know that it means: $((-(5) \cdot 2) + (10/5)) - 8$ because the operators of arithmetic have the implicit order of precedence

–> decreasing order –>

Do 1st <————-> Do last

$$-, \quad \begin{array}{c} \cdot \\ / \end{array}, \quad \begin{array}{c} + \\ - \end{array}$$

Precedence of Logical Connectives

We say that operators with a higher order of precedence “have a tighter binding”.

Similarly for logical connectives we define the order of precedence as:

Do 1st <————————-> Do last

$$\neg, \begin{matrix} \wedge \\ \vee \end{matrix}, \begin{matrix} \longrightarrow \\ \leftrightarrow \end{matrix}$$

Thus $((p \wedge \neg(q)) \rightarrow r)$ becomes: $p \wedge \neg q \rightarrow r$

Properties of Binary Operators

Def: A binary operator $* : \mathcal{V}^2 \to \mathcal{V}$ is *commutative* if for all values of $p, q \in \mathcal{V}$:

$$p * q = q * p$$

E.g. Addition and multiplication are commutative over the reals but division is not.

$\wedge, \vee, \leftrightarrow$ are commutative

but $\to$ is not!

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

Properties of Binary Operators

Def: A binary operator $*: \mathcal{V}^2 \to \mathcal{V}$ is *associative* if for all values of $p, q, r \in \mathcal{V}$:

$$(p * q) * r = p * (q * r)$$

E.g. $+$ and $\cdot$ are associative over the reals but $/$ is not (e.g. $(4/2)/2 = 1$ but $4/(2/2) = 4$).

$\wedge, \vee, \leftrightarrow$ are associative. Therefore $(p \wedge q) \wedge r$ and $p \wedge (q \wedge r)$ "mean the same thing" so we write $p \wedge q \wedge r$.

(Similar to writing $5 \cdot 2 \cdot 4$ for integer mult.)

Note: $(p \wedge q) \vee r$ is NOT "equivalent" to $p \wedge (q \vee r)$! (Check using truth tables.)

$\rightarrow$ is not associative!

	p	q	r	$(p \rightarrow q)$	$\rightarrow r$	$p \rightarrow$	$(q \rightarrow r)$
0	F	F	F	T	F	T	T
1	F	F	T	T			T
2	F	T	F	T			F
3	F	T	T	T			
4	T	F	F				
5	T	F	T				
6	T	T	F				
7	T	T	T				

Row 0 of the truth table provides counter example so we can stop.

Note that there are 2^3 rows numbered 0 to $7 = 2^3 - 1$.

In general, a truth table for compound proposition will have 2^n rows, where $n=$ number of unique propositional variables occuring in the expression.

Count in binary with F being 0 and T being 1 to cover all cases.

Tautologies and Contradictions

Def: A logical expression is a *tautology* (*contradiction*) if it is true (false) under all possible assignments to its propositional variables.

E.g. $p \vee \neg p$ is a tautology since its truth table results in all T's while $p \wedge \neg p$ is a contradiction:

p	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
F	T	T	F
T	F	T	F

The negation of any tautology is a contradiction and vice versa. Why?

If S is a tautology, then so is any substitution instance of it (i.e. consistently replacing variables with any other formulas results in a tautology!).

E.g $(p \rightarrow q) \vee \neg(p \rightarrow q)$ is a tautology.

Logical (Semantic) Equivalence

Def: Two propositional formulas are *logically equivalent* if they have the same truth table.

This means the propositions define the same function from $\mathcal{V}^n$ to $\mathcal{V}$ where $n :=$ number of propositional variables in the formulas.

E.g. The formulas $\neg(p \wedge q)$ and $\neg p \vee \neg q$ define the same function $f : \mathcal{V}^2 \to \mathcal{V}$

p	q	$\neg(p \wedge q)$	$\neg p \vee \neg q$
F	F	T	T
F	T	T	T
T	F	T	T
T	T	F	F

Logical (Semantic) Equivalence (cont)

Note that $\neg(p \wedge q)$ and $\neg p \vee \neg q$ are logically equivalent iff $\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$ is a tautology. Why?

p	q	$\neg(p \wedge q)$	$\neg p \vee \neg q$	$\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$
F	F	T	T	
F	T	T	T	
T	F	T	T	
T	T	F	F	

This is why Rubin refers to logical equivalence as *tautological equivalence* and when ϕ is logically equivalent to ψ writes: $\phi \Leftrightarrow \psi$.

Huth+Ryan refer logical equivalence as *semantic equivalence* and write: $\phi \equiv \psi$. It all means the same thing. The formulas have the same truth table.

Building the World with NAND

NAND: (Negation of AND)

$\text{NAND} : \mathcal{V}^2 \rightarrow \mathcal{V}$

p	q	$p \,\text{NAND}\, q$	$\neg(p \wedge q)$
F	F	T	T
F	T	T	T
T	F	T	T
T	T	F	F

Thus $p \,\text{NAND}\, q \equiv \neg(p \wedge q)$
"$p \,\text{NAND}\, q$ is logically equivalent to $\neg(p \wedge q)$"

$\neg p \equiv T \,\mathrm{NAND}\, p$

p	$\neg p$	$T \,\mathrm{NAND}\, p$
F	T	T
T	F	F

This means NAND can implement negation!

Note: Using T and F in the formulas is a minor abuse of notation! It *is* possible to “fake” $\neg p$ without using T or F. How?

$p \wedge q \equiv \neg(p \,\mathrm{NAND}\, q)$

p	q	$p \wedge q$	$p \,\mathrm{NAND}\, q$	$\neg(p \,\mathrm{NAND}\, q)$
F	F	F	T	F
F	T	F	T	F
T	F	F	T	F
T	T	T	F	T

So

$$\begin{aligned} p \wedge q &\equiv \neg(p \,\mathrm{NAND}\, q) \\ &\equiv (T \,\mathrm{NAND}\, (p \,\mathrm{NAND}\, q)) \end{aligned}$$

Also $p \vee q \equiv \neg p \,\mathrm{NAND}\, \neg q$ and similar NAND-only equivalents exist for $\rightarrow$ and $\leftrightarrow$.

Any logical formula uses a combination of $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$

Therefore any logic formula can be written as an equivalent formula using only NAND .

Note: This is an informal proof. To do it rigorously we have to use structural induction on propositional formulas.

Normal Forms

Normal forms in mathematics are canonical representations (i.e. all equivalent objects result in the same representation).

Def: A formula ϕ with $p_1, p_2, \ldots p_n$ propositional variables is in *Disjunctive Normal Form* (DNF) if it is has the structure:

$$(x_1^1 \wedge x_2^1 \wedge \ldots \wedge x_n^1) \vee \ldots \vee (x_1^m \wedge x_2^m \wedge \ldots \wedge x_n^m)$$

where $m \leq 2^n$ and for $i = 1, \ldots n$ and
$j = 1, \ldots m, \quad x_i^j$ is either p_i or $\neg p_i$

E.g. $(\neg p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge \neg r)$ is in DNF
$\neg(p \vee q) \wedge r$ is not. Each of the series of conjunctions picks out a row of the truth table where formula is true. DNF ORs together the ANDs for the true rows.

Normal Forms (cont)

Consider the truth tables for the formulas $\neg p \land \neg q \land r$ and $p \land \neg q \land \neg r$:

	p	q	r	$\neg p \land \neg q \land r$	$\neg p \land q \land r$
0	F	F	F	F	F
1	F	F	T	T	F
2	F	T	F	F	F
3	F	T	T	F	T
4	T	F	F	F	
5	T	F	T	F	
6	T	T	F	F	
7	T	T	T	F	

For $\neg p \land \neg q \land r$ only row 1 is true.

For $\neg p \wedge q \wedge r$ only row 3 is true.
What conjunct is only true on row 6?

$(\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r) \vee (p \wedge q \wedge \neg r)$ is true on rows 1, 3 & 6.
Why?

Theorem: For every truth table, there is a propositional formula that generates the truth table.

Normal Forms (cont)

Theorem: Every propositional formula that is not a contradiction is a logically equivalent to a DNF formula.

Corollary: For ϕ, ψ not contradictions, $\phi \equiv \psi$ iff ϕ and ψ have the same DNF representation.

Proof: Two formulas are logically equivalent if and only if they have the same truth table (i.e. same true rows) & thus the same DNF.

Application: Minimizing gate delays

If each input & its negation are available, any logic function can be implemented with one "stage" of multi-input AND gates followed by one "stage" of multi-input OR gates.

Logical Implication

Def: We say ϕ *logically implies* ψ if $\phi \to \psi$ is a tautology. In this case Rubin writes $\phi \Rightarrow \psi$. If ϕ is a conjunction (i.e. ϕ is $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$) then we say $\phi_1, \phi_2, \ldots, \phi_n$ logically imply ψ.

Huth+Ryan write $\models \phi \to \psi$ or $\phi \models \psi$.

Premises $\phi_1, \ldots, \phi_n$ with conclusion ψ is a *sound* or *valid argument*, denoted

$$\phi_1, \phi_2, \ldots, \phi_n \models \psi$$

if whenever all the ϕ_is are true, then ψ is true.

Theorem: $\models \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n \to \psi$ if and only if $\phi_1, \phi_2, \ldots, \phi_n \models \psi$.

Modus Ponens: $p, p \rightarrow q \models q$

p	q	$p \rightarrow q$	$p \wedge (p \rightarrow q)$	q
F	F	T	F	F
F	T	T	F	T
T	F	F	F	F
T	T	T	T	T

Checking validity (soundness) of arguments:

- To prove an argument is valid we only have to check that the conclusion (ψ) is true in rows in which all the premises (ϕ_i's) are true.
- To prove an argument is *invalid* (unsound), we need only find one counter example, a row in which each ϕ_i is true but ψ is false.

Examples: 1. $(p \to q) \to r \models p \to (q \to r)$ but
$p \to (q \to r) \not\models (p \to q) \to r$

2. $p, p \rightarrow q, \neg r \rightarrow \neg q \models r$

	p	q	r	$p \rightarrow q$	$\neg r \rightarrow \neg q$	r
0	F	F	F			
1	F	F	T			
2	F	T	F			
3	F	T	T			
4	T	F	F	F		
5	T	F	T	F		
6	T	T	F	T	F	
7	T	T	T	T	T	T

Special Cases

1. **No premises:** Premises restrict the cases that we have to consider. No premises means we consider all cases. ψ is a valid argument by itself if it is always true (i.e. it is a tautology). Then we write $\models \psi$ and say that ψ is *valid*.

2. **Premises never all true:** At least one ϕ_i is always false so $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$ is a contradiction. Then $\phi_1, \ldots, \phi_n \models \psi$.

"If pigs could fly then I'd enjoy brussel sprouts!"
p : Pigs fly; b : Enjoy sprouts

This ($p \models b$) is an *invalid argument.* Why use it?

The real argument is:

$$p, \neg p \models b$$

which *is* a valid argument.

Why is it valid? There is no counter example where $p \wedge \neg p$ is true and b is false.

Ex falso quod libet! i.e. "From false all things are possible!"

$\neg p$ is an *implicit assumption* in the verbal argument. Implicit assumptions are <u>extremely dangerous</u> in software. Make your assumptions explicit!

How do you make software assumptions explicit? Documentation, using strongly typed languages, dependent typing in PVS, *etc.* ...

Validity & Satisfiability

Let ϕ be some formula of propositional logic. In the case that $\models \phi$, we say that ϕ is *valid*.

In the case that ϕ is **not** valid (i.e., there is some assignment to its variables that makes it false) we will write $\not\models \phi$.

If there is some assignment to the propositional variables that makes ϕ true (i.e., there is one or more T in the final column of ϕ's truth table), then we say that ϕ is *satisfiable*.

Proposition: ϕ is satisfiable iff $\not\models \neg\phi$.

Conjunctive Normal Form

Def: A formula with $p_1, p_2, \ldots p_n$ propositional variables is in *Conjunctive Normal Form* (CNF) if it is has the structure:

$$(x_1^1 \vee x_2^1 \vee \ldots \vee x_n^1) \wedge \ldots \wedge (x_1^m \vee x_2^m \vee \ldots \vee x_n^m)$$

where $m \leq 2^n$ and for $i = 1, \ldots n$ and
$j = 1, \ldots m, \quad x_i^j$ is either p_i or $\neg p_i$

E.g. $(\neg p \vee \neg q \vee r) \wedge (p \vee \neg q \vee \neg r)$ is in CNF
$\neg(p \wedge q) \vee r$ is not. Each of the series of disjunctions rules out a row of the truth table where formula is false. CNF ANDs together the ORs for the false rows.

One way to obtain the CNF form of a formula ϕ is to write down the DNF for $\neg\phi$ and then negate it and "Demorgan it to death".

Using CNF to Check $\models \phi$

Q: CNF seems a little harder to understand than DNF, so why use it?

A: Because it is trivial to check $\models \phi$ if ϕ is in CNF.

Why? Because

$$\begin{aligned} \models \quad & (x_1^1 \vee x_2^1 \vee \ldots \vee x_n^1) \wedge (x_1^2 \vee x_2^2 \vee \ldots \vee x_n^2) \\ & \ldots \wedge (x_1^m \vee x_2^m \vee \ldots \vee x_n^m) \end{aligned}$$

if and only if

$$\begin{aligned}
\models \quad & (x_1^1 \vee x_2^1 \vee \ldots \vee x_n^1) \\
& \text{and} \\
\models \quad & (x_1^2 \vee x_2^2 \vee \ldots \vee x_n^2) \\
& \vdots \\
& \text{and} \\
\models \quad & (x_1^m \vee x_2^m \vee \ldots \vee x_n^m)
\end{aligned}$$

If each x_i^j is a *literal* (e.g., p) or its negation (e.g., $\neg p$) then $\models (x_1^j \vee x_2^j \vee \ldots \vee x_n^j)$ iff there exists k, l s.t. $x_k^j = p$ and $x_l^j = \neg p$.