Week 2 - Logic

Well Formed Formulas (wff): A complete expression, i.e singular propositional letter, or enough variables in an expression.

Notation: Appearing in order of their bind tightness. (Allows us to drop parentheses)

Not ($\neg$) : Negative that precedes a propositional letter, opposite of letter value

And ($\wedge$) : Conjunction, **t** if both elements are **t**

Or ($\vee$) : Disjunction, **t** if at least one element is **t**

XOR ($\oplus$) : Exclusive Or, **t** if both elements are different

If Then ($\Rightarrow$) : Implication, *if* A is **t** *then* return B, else **t**

Iff ($\Leftrightarrow$) : If and only if, **t** if both elements are the same

Truth Table:

A	B	$\neg$A	A $\wedge$ B	A $\vee$ B	A $\Rightarrow$ B	A $\Leftrightarrow$ B	A $\oplus$ B
f	f	t	f	f	t	t	f
f	t	t	f	t	t	f	t
t	f	f	f	t	f	f	t
t	t	f	t	t	t	t	f

Nor ($\downarrow$): Not or, opposite of or

Nand ($\uparrow$): Not and, opposite of and

Tenary Connectives: Connectives with 3 elements

If Then Else: If A then B else C, if A is **t**, then return B, else return C

Median: Median(A,B,C), chooses the majority of the 3 elements

Validity: A formula is *valid* if no truth assignment makes it false (always **t**). Else *non-valid*.

Truth Assignment: A combination of values we can assign to a formula. For example $\theta = \{$A: **t** , B: **f**$\}$

Satisfiability: A formula is *unsatisfiable* if no truth assignment makes it true (always **f**). Otherwise it is *satisfiable*.

Tautology: A valid propositional formula. Means that it is never false.

Contradiction: An unsatisfiable propositional formula. Means that it is never true

Vacuous: A valid statement can mean a void of information. For example A$\Rightarrow$A is always valid, and a vacuous statement as it provides no extra information other than if A is true, it is true.

Substitution: We can preserve validity by *substitution* of propositional letters with formulas or expressions. All occurrences of the letter should be replaced by the same formula. Substitution also preserves unsatisfiability.

Model: If A makes F true, then A is a model of F.

Logical Consequence ($\models$): G is a *logical consquence* of F iff every model of F is a model of G. In other words, everything that makes F true must also make G true.

F and G have the same truth cases
{Models of F} $\subseteq$ {Models of G}
Notation: F $\models$ G

Logical Equivalence ($\equiv$): If F $\models$ G, and G $\models$ F then F and G are logically *equivalent*

F and G have the same logic table values
{Models of F} = {Models of G}
Notation: F $\equiv$ G

Equivalence: Preserves logical equivalence, logical consequence, validity

Absorption: Anything with itself is still itself

P and P equals P, P or P equals P
$P \wedge P \equiv P$
$P \vee P \equiv P$

Communtativity: Order doesn't matter

$P \wedge Q \equiv Q \wedge P$
$P \wedge Q \equiv Q \vee P$

Associativity: Brackets don't matter

$P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$
$P \vee (Q \vee R) \equiv (P \vee Q) \vee R$

Distributivity: Can expand the brackets

$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$
$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$

Double Negation: Two negatives makes a positive

$P \equiv \neg\neg P$

De Morgan: Not(A something B) = Not A (opposite something) Not B

$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$

Implication: If P is **f**, then result is **t** or $\neg$P. Otherwise $\neg$P is **f** and the result is determined by Q

$P \Rightarrow Q \equiv \neg P \vee Q$

Contraposition: Reverse equation and swap signs on If Then expressions

$\neg P \Rightarrow \neg Q \equiv Q \Rightarrow P$
$P \Rightarrow \neg Q \equiv Q \Rightarrow \neg P$
$\neg P \Rightarrow Q \equiv \neg Q \Rightarrow P$

Biimplication: **t** if both P and Q are true, or both false

$P \Leftrightarrow Q \equiv (P \wedge Q) \vee (\neg P \wedge \neg Q)$

Last Equivalences:

($\bot$): Any unsatisfiable formula

($\top$): Any valid formula

Duality: Not unsatisfiable/valid equals opposite

$\neg\top \equiv \bot$
$\neg\bot \equiv \top$

Negation from Absurdity: If P then it's unsatisfiable is the same as not P

$P \Rightarrow \bot \equiv \neg\, P$

Identity:

$P \vee \bot \equiv P$
$P \wedge \top \equiv P$

Dominance:

$P \wedge \bot \equiv \bot$
$P \vee \top \equiv \top$

Contradiction: P and Not P is always unsatisfiable (**f**)

$P \wedge \neg P \equiv \bot$

Excluded Middle: P or Not P is always valid (**t**)

$P \vee \neg P \equiv \top$

P	$\neg$P	P $\wedge$ $\neg$P	P $\vee$ $\neg$P
0	1	0	1
1	0	0	1

Week 3 - Symbolic Deduction and Predicate Logic

Literal: Describes a propoisitonal letter or its negation.

Conjunctive Normal Form (CNF): It is a conjunction of clauses, where the clauses are disjunctions of literals.

Eg. $(A \vee \neg B) \wedge (B \vee C \vee D) \wedge A$
Note: The brackets or a single literal is considered a clause

Disjunctive Normal Form (DNF): A disjunction of clauses, where the clauses are conjunction of literals

Eg. $(\neg A \wedge \neg B) \vee (\neg B \wedge C) \vee (A \wedge \neg D)$

Conversion: Every propositional formula can be expressed in both CNF or DNF. The steps to do so are as follows.

1. Eliminate all occurences of $\oplus$, using
   $A \oplus B \equiv (A \wedge B) \vee (\neg A \vee \neg B)$
2. Eliminate all occurences of $\Leftrightarrow$, using
   $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow B)$
3. Eliminate all occurences of $\Rightarrow$, using
   $A \Rightarrow B \equiv \neg A \vee B$
4. Use DeMorgan's Laws to push $\neg$ inward over $\wedge$ and $\vee$
   $\neg(A \wedge B) \equiv \neg A \vee \neg B$
   $\neg(A \vee B) \equiv \neg A \wedge \neg B$
5. Eliminate double negations using $\neg\neg A \equiv A$
6. Use the distributive laws to get the final required form (CNF or DNF)

Reduced CNF: A CNF formula is in reduced CNF (RCNF) if, for each of its clauses, no propositional letter occurs twice.

Eg. $(A \vee \neg B \vee \neg A) \wedge (\neg C \vee \neg B) \wedge (C \vee \neg A \vee C \vee B)$
$\equiv (\neg C \vee \neg B) \wedge (C \vee \neg A \vee B)$

Canonical Form: If a normal form leads to a unique representation for every Boolean function. (CNF and DNF are not always unique)

XOR Normal Form: A canonical form that presents the function in a 'sum of products' form, using 'exclusive or' and conjunction.

Eg. $(A \Rightarrow B) \wedge (B \oplus C) \equiv ABC \oplus AC \oplus B \oplus C$

- $P \wedge Q$ abbreviates to PQ
- For a formula $A_1 \oplus A_2 ... \oplus A_n$, it will be true if an odd number of $\{A_1, A_2, An\}$ is true.
- $(P \wedge Q) \oplus Q \equiv (P \wedge Q) \oplus (Q \wedge Q) \equiv PQ \oplus Q$

Binary Decision Diagrams (BDD): Provides another canonical form through graph representation. Each literal is represented as a node, that results in either a **t** or **f** outcome. If there is only one outcome, we can easily determine if it is valid or unsatisfiable.

Clause: A set (disjunction) of literals

Clausal Form: Writing a formula in CNF using sets

Eg. $(P \vee \neg Q \vee S) \wedge (P \vee \neg R \vee S) \wedge (\neg S \vee \neg P) \wedge (\neg S \vee Q \vee R)$

Can be expressed as:
$\{\{P, S, \neg Q\}, \{P, S, \neg R\}, \{\neg P, \neg S\}, \{Q, R, \neg S\}\}$

Empty Clauses (Ø): The natural reading is dependent on the conjunction or disjunction.

If it is a disjunction, the natural reading is **f**, as **f** is a neutral element for $\vee$, as $\mathbf{f} \vee A \equiv A$.
Therefore Ø would be represented as $\perp$

If it is a conjunction, the natural reading is **t**, **t** is a neutral element for $\wedge$.
Therefore Ø is represented as $\top$

For CNF:
The set of Ø of clauses is valid. (Don't need to do anything to satisfy)
Any set {Ø, ...} of clauses is unsatisfiable. (Can never satisfy an empty clause in a set)

Resolvent: A clause that is a logical consequence of two original clauses.

Eg. Consider the two clauses $\neg P \vee A$ and $P \vee B$
If $P = \mathbf{t}$, it reduces to A and **t**
If $P = \mathbf{f}$, it reduces to **t** and B
We get the clause $A \vee B$ as a logical consequence of the two clauses.

Resolution Refutation: Through resolving clauses, we obtain a deduction of $\perp$ from S. Hence we can conclude that the formula is unsatisfiable.

Resolution Deduction: If we have clause C from a set S of clauses, there is a finite sequence $C_1, C_2 ... C_n$ of clauses such that $C_n = C$ and for each member i, C_i is either a member of S or a resolvent of $C_{members}$

Using Refutations:

Proving Validity: Prove F is valid.

1. Put $\neg F$ in RCNF, yielding a set S of clauses.
2. Refute S, or deduce $\perp$ from S.

From this, if the negation of F is unsatisfiable, then F is a tautology and valid.

Proving that it satisfies a condition: Show $F \models G$, ie, F satisfies some property G.

We know that $F \models G$ iff $F \wedge \neg G$ is unsatisfiable

1. Negate G and bring into RCNF
2. Add those clauses to the set F
3. Find a refutation of the resulting set of clauses.
4. If we derive an empty clause or $\perp$, then $F \models G$

Predicate Logic: Allows use to finitely express statements that deal with infinite collections of objects, and express relations, transitive verbs and relative pronouns.

Predicate: A function that maps individuals to **t** or **f**, similar to a propositional letter in propositional logic. Denoted with uppercase symbols.

Variables: Ranges over collections. Denoted with lowercase letters, and tend to be at the end of the alphabet. Constants are at the start.

Functions: Can take variables as inputs. Unlike predicates, does not return any value and we have the underlying assumption that it is true. Also denoted with lowecase letters.

Quantifiers: Specifies how many of a variable.

Existential Quantification (∃): There exists, a generalised or infinite ∨.
(If at least one is true, then it is true)

Universal Quantification (∀): For all, a generalised or infinite ∧.
(All must be true to be true)

Arity: A number that says how many arguments the function takes. Each predicate symbol comes with an arity.

Term: Either a variable, a constant or a construction f($t_1 ... t_n$), where f is a function symbol of arity n, and each t is a term. Considered individual and an object.

Atomic Formula (Atom): A construction P($t_1 ... tn$) where P is a predicate symbol of arity n, and each t is a term. Is an assertion, and can be either true or false.

Literal: An atomic formula or its negation.

Bound: A variable which is in the scope of a quantifier, and binds that variable.

Free: If a variable is not bound, then it is free.

Closed: A formular with no free variable occurences.

Week 4 - Predicate Logic, Semantics and Clausal

Interpretation: Also known as a structure. Consists of several parts:

- A non-empty set D (the domain or universe)
- An assignment, to each n-ary predicate symbol P of an n-place function $\mathbf{p} : D^n \to \{\mathbf{f}, \mathbf{t}\}$
- An assignment, to each n-ary function symbol g, of an n-place function $\mathbf{g}$: $D^n \to D$
- An assignment to each constant a of some fixed element of D

Formulas with free variables: To give meaning to formulas with free variables, we need a valuation and an interpretation.

Valuation: $\sigma : var \to D$ for free variables
Interpretation: a way of interpreting atomic formulas and mathematical symbols (eg. $>$)
Note: connectives are given their usual meaning/interpretation

Given an intepretation I, we can get a valuation function from terms automatically:

$$\sigma(a) = d$$
$$\sigma(g(t_1, ..., t_n)) = \mathbf{g}(\sigma(t_1), ..., \sigma(t_n))$$

where d is the element of D that I assigns to a, and $\mathbf{g} : D^n \to D$ is the function that I assigns to g

Truth of a Formula:

If a formula is closed, then the truth of that formula depends only on the given interpretation. If the formula has free variables, then we want to define the truth of a formula compositionally.

Making a Formula True: Given an interpretation I with domain D and a valuation σ

- σ makes $P(t_1, ..., t_n)$ true iff $\mathbf{p}(\sigma(t_1, ..., \sigma(t_n))) = \mathbf{t}$, where $\mathbf{p}$ is the meaning that I gives P
  i.e If σ makes every instance of p true, then the equation is true.
- σ makes $\neg$F true iff σ does not make F true
  i.e If σ makes F false, then it must also make its negation true.
- σ makes $F_1 \wedge F_2$ true iff σ makes both of F_1 and F_2 true
  i.e Both statements must be made true by σ, similarily, if it was $\vee$, then at least one of F_1, F_2 needs to be true
- σ makes $\forall$x F true iff $\sigma_{x \to d}$ makes F true for every d $\in$ D
  i.e For every value of x, where x is bound by D, sigma is true.

We can also define:

$$\exists \text{x } F \equiv \neg \forall \text{x } \neg F$$

Which the meaning of every other formula follows from.

True in Interpretation: A wff F is true in interpretation iff every valuation makes F true (for I). If it is not true, then it is false in interpretation.

Model: A model for F is in interpretation I such that F is true in I.
We write this as $I \models$ F

Logically Valid: A wff F is logically valid iff every interpretation is a model for F.
We write this as $\models$ F

Logical Consequence: F_2 is a logical consequence of F_1 iff $I \models F_2$ whenever $I \models F_1$
We write this as $F_1 \models F_2$

Logical Equivalence: F_1 and F_2 are logically equivalent iff $F_1 \models F_2$ and $F_2 \models F_1$.
We write this as $F_1 \equiv F_2$

Closed Formula: A close wff F is

- satisfiable iff $I \models$ F for some interpretation I
  - find one example where it is true
- valid iff $I \models$ F for every interpretation I
- unsastisfiable iff $I \not\models$ F for every intepretation I
- non-valid iff $I \not\models$ F for some interpretation I
  - find one example where it is false
- F is valid iff $\neg$F is unsatisfiable
- F is non-valid iff $\neg$ F is satisfiable

Order of Quantifiers: The order of different quantifiers changes the meaning. If the quantifiers are the same, then the meaning stays the same.

$\forall$x $\exists$y means for each x, there exists a y that satisfies something
$\exists$y $\forall$x means that there exists some y that satisfies something for every x

Quantifier Rules:

We cannot 'push quantifiers in', however we can rearrange some formulas.

$$\begin{aligned}
\exists \text{x } (\neg F_1) &\equiv \neg \forall \text{x } F_1 \\
\forall \text{x } (\neg F_1) &\equiv \neg \exists \text{x } F_1 \\
\exists \text{x } (F_1 \vee F_2) &\equiv (\exists \text{x } F_1) \vee (\exists \text{x } F_2) \\
\forall \text{x } (F_1 \wedge F_2) &\equiv (\forall \text{x } F_1) \wedge (\forall \text{x } F_2) \\
\exists \text{x } (F_1 \Rightarrow F_2) &\equiv (\forall \text{x } F_1) \Rightarrow (\exists \text{ x} F_2)
\end{aligned}$$

If G is a formula with no free occurences of x, then we get the following formulas, regardless of F. F may have free occurences of x.

$$\begin{aligned}
\exists \text{x G} &\equiv \text{G} \\
\forall \text{x G} &\equiv \text{G} \\
\exists \text{x}(\text{F} \wedge \text{G}) &\equiv (\exists \text{x F}) \wedge \text{G} \\
\forall \text{x}(\text{F} \vee \text{G}) &\equiv (\forall \text{x F}) \vee \text{G} \\
\forall \text{x}(\text{F} \Rightarrow \text{G}) &\equiv (\exists \text{x F}) \Rightarrow \text{G} \\
\forall \text{x}(\text{G} \Rightarrow \text{F}) &\equiv \text{G} \Rightarrow (\forall \text{x F})
\end{aligned}$$

Skolemization: A process that eliminates existential quantifiers so we can resolve them.

Skolem Constant:

Consider F = $\exists$x $\forall$y P(x,y) under some interpretation I

F is satisfiable iff some valuation σ makes $\forall$ y P(x,y) true.
Say that σ, with σ(x) = d_0 makes $\forall$ y P(x,y) true.

We can replace this x with a fresh constant a, giving: $\forall$y P(a, y)
This formula is satisfiable iff F is.
If I satisfies F, then a can be mapped to d_0, hence the formula with constant a is also satisfiable.
If the formula is unsatisfiable, then there is no valuation of x that will make the formula true.
Therefore the formula with the constant a is also unsatisfiable.

We can then conclude that replacing x with a makes it equisatisfiable, but are not equivalent.

Skolem Function:

Consider G = $\forall$y $\exists$x P(x,y)
We cannot replacex with a constant, as x is a function of the value of y.

Instead, we replace x with f(y), giving: $\forall$ y P(f(y), y)

Using similar theory, we can conclude that these formulas are equisatisfiable.

Converting to Clausal Form

1. Replace occurences of $\oplus$, $\Leftrightarrow$, $\Rightarrow$
2. Drive negation in
3. Standardise bound variables apart
4. Eliminate existential quantifiers (Skolemize)
5. Eliminate universal quantifiers (just remove/ignore them)
6. Bring to CNF (using distributive laws)

Week 5 - Unification, Resolution and Induction

Substitution: A finite set of replacements of variables by terms.
eg. $\theta = \{x_1 \mapsto t_1, x_2 \mapsto t_2, ..., x_n \mapsto t_n\}$, where x_i are variables and t_i are terms.
All these variables are replaced by terms simultaneously.

Unifier: A *unifier* of two terms s and t is a substituion θ such that $\theta(s) = \theta(t)$

Unifiable: Two terms are unifiable iff there exists a unifier for s and t

Most General Unifier (mgu): The mgu for s and t is a substituion such that

1. θ is a unifier for s and t
2. Every other unifier σ of s and t can be expressed as $\tau \circ \theta$ for some substituion τ
   Mgu is the unifier with the least steps, and every other unifier can be expressed as some form of the mgu.

If s and t are unifiable, then they have a most general unifier.

Unifier Rules:

- A variable x can be mapped to anything
- A constant a cannot be mapped
- A function *f(x)* cannot be mapped to a constant.
- A function *f(x)* cannot be mapped to x, the terms must be finite.

Unification Algorithm:

Input: Two terms s and t
Output: If they are unifiable, then we will get the mgu. Otherwise it will return failure.
Algorithm: Start with the set of equations s = t. This is the *singleton* set, with one element.

In the following cases, one of the equations in the set has this form, then perform an action on it.

1. $F(s_1, ..., s_n) = F(t_1, ..., t_n)$:
   - Replace the equation by the n equations $s_1 = t_1, ..., s_n = t_n$
2. $F(s_1, ..., s_n) = G(t_1, ..., t_m)$ where $F \neq G$ or $n \neq m$:
   - Halt and return failure.
3. $x = x$:
   - Delete equation, is valid but gives us no information.
4. $t = x$ where t is not a variable:
   - Rearrange the equation to $x = t$, as we prefer variables on the LHS.
5. $x = t$ where $t \neq x$, but x occurs in t:
   - Halt and return failure. Cannot map *f(x)* to x
6. $x = t$ where t contains no x, but x occurs in other equations:
   - Replace x by t in those other equations.

Normal Form: If the algorithm halts without returning failure, then the term equation system is left in a normal form.

- All variables on the LHS are different
- No variables are on the RHS if they appear on the LHS. Should either be non-variables, or fresh ones.

Resolvents:

- Two literals L and $\neg L'$ are complementary if $\{L,\ L'\}$ is unifiable.
- Let C_1 and C_2 be clasues, renamed apart.
  Let θ be an mgu of complementary literals $\{L, \neg L'\}$ with L a literal in C_1 and $\neg L'$ a literal in C_2.
  Then the resolvent of C_1 and C_2 is the union $\theta(C_1\ \{L\}) \cup (C_2\ \{\neg L'\})$

Refutation': To resolve something, we follow a similar process to predicate logic, but through mapping. We negate the final statement that is supposed to be the result of the other arguments. If we recieve an empty clause, the input is unsatisfiable, hence the final statement follows from the others.

Factoring: Rewriting clauses if two terms in a claus can be mapped to each other.

Induction: Given a statement, prove it using induction

1. Basis Step: Show S(0), i.e the base case is valid.
2. Inductive Step: Take S(n) as the induction hypothesis and use it to establish S(n+1)

Course of Values Induction: Generalising induction.
To prove a claim P(n), we can take the conjunction of terms P(0) $\wedge$ P(1) $\wedge$ ... P(n-1) as the induction hypothesis.

Structural Induction: Using the structure of a recursively defined object to prove a statement. Useful for Trees.