Contents

Unit 1 Propositional Logic

Week 1 Formalizing Logic

1.1 Intro

Definition. Syllogism: Logical argument of two propositions and a conclusion.

Definition. Proposition: A statement that is either true or false. Commonly we use 1 to mean true and 0 to mean false.

Definition. Hypothetical Syllogism: A form of valid syllogism: if p then q, if q then r concludes if p then r.

Definition. Disjunctive Syllogism: A form of valid syllogism: p or q not q, concludes p.

Definition. Modus Ponens: A form of valid syllogism: if p then q, p, concludes q.

Definition. Propositional Variable: A variable representing a proposition, i.e. it is either true (1) or false (0). It is also called an atomic proposition in that it cannot be further reduced or subdivided.

Definition. Compound Proposition: A proposition obtained by combining several atomic propositions.

Definition. Negation: Where p is a proposition, $\neg p$ is the compound proposition "not p" which is true when p is false and false when p is true.

Example: The truth table for a negation follows as:

p	$\neg p$
1	0
0	1

Definition. Conjunction: Where p and q are propositions, $p \wedge q$ is the compound proposition "p and q" which is true when both p and q are true and false otherwise.

Definition. Disjunction: Where p and q are propositions, $p \vee q$ is the compound proposition "p or q" which is true when p or q (or both) are true and false only when both p and q are false (note this is the inclusive or).

Definition. Implication: Where p and q are propositions, $p \implies q$ is the compound proposition "if p, then q" which is true except when p is false and q is true.

p is called the antecedent and q is called the consequent. The truth table for implication is

p	q	$p \implies q$
1	1	1
1	0	0
0	1	1
0	0	1

Definition. Equivalence: Where p and q are propositions, $p \iff q$ is the compound proposition "p if and only if q" which is true only when both p and q are true, or both are false. Often abbreviated "p iff q".

Remark. Ambiguity vs. Imprecision: A sentence is ambiguous if it has more than one distinct meanings, whereas it is imprecise if the circumstances under which it is true are not obvious.

1.2 Propositional Syntax

Definition. $\mathcal{L}^p$: The formal language of propositional logic. Strings in $\mathcal{L}^p$ comprise of three classes of symbols:

- Propositional symbols: $p, q, \dots$
- Connective symbols: $\neg, \wedge, \vee, \implies, \iff$
- Punctuation symbols: $($, $)$

Definition. Expression: An expression is a finite string of the above symbols. It is defined to have a length equal to the number of symbols. The expression of the length zero is defined to be ε, the empty expression.

Two expressions are said to be equal if they have the same length and the same symbols in the same order. In this case we say expressions U, V are equal by $U = V$. Scanning proceeds as usual from left to right.

Note that representing expressions by variables does not use symbols, but rather meta-symbols. Similarly, $=$, $\neq$, etc. are meta-symbols.

Definition. Concatenation: The concatenation of two expressions U and V is denoted UV and is the elements of U followed by those of V. Note $\varepsilon U = U = U\varepsilon$.

Definition. Segment: If $U = W_1 V W_2$ where U, V, W_1, W_2 are expressions then V is a segment of U, and if $V \neq U$ then V is a proper segment of U. Note every expression is a segment of itself and ε is a segment of every expression. If $U = VW$, we further specify V is the initial segment or prefix of U and W is the terminal segment of suffix of U. Similarly, we can add the proper prefix to each of these.

Definition. Atomic Formula: (or atom) An expression of length one in $\mathcal{L}^p$. The set of atomic formulas is denotes $Atom(\mathcal{L}^p)$.

Definition. $Form(\mathcal{L}^p)$: The set of formulas of $\mathcal{L}^p$, defined recursively by the base case where it contains all atoms and then if $A, B \in Form(\mathcal{L}^p)$ then $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \implies B)$, $(A \iff B)$. These 5 elements are the formation rules in $\mathcal{L}^p$.

Theorem. Unique Readability Theorem: Every formula $A \in Form(\mathcal{L}^p)$ is exactly one of an atom, $(\neg B)$, $(B \wedge C)$, $(B \vee C)$, $(B \implies C)$, or $(B \iff C)$. In each case it is of that form in exactly one way.
Proof. To prove this theorem, prove that: (a) the first symbol of A is either a (or a propositional symbol (atom). (b) A has an equal number of left and right parentheses. (c) every

non-empty proper initial segment of A has more (than). (d) A has a unique construction as a formula. Prove this by performing structural induction on the number of the connectives in A. $\square$

Definition. Precedence Rules: Logical operators have the following order of precedence:

1. $\neg$
2. $\wedge$
3. $\vee$
4. $\implies$
5. $\iff$

Definition. Scope: If $(\neg A)$ is a segment of a formula C, then A is the scope of the negation in C. If $(A \star B)$ is a segment of a formula C for some binary operator $\star$, then A is called the left scope of the operation (e.g. conjunction) and B is called the right scope.

Week 2 Proofs

2.1 Propositional Logic Semantics

Definition. Syntax: Syntax is concerned with the rules for constructing formulas (as seen in computer science) If a formula is in $\mathcal{L}^p$ but not $Form(\mathcal{L}^p)$, it corresponds to a syntax error.

Definition. Semantics: Semantics is concerned with meaning. In propositional logic, this is either true or false. Atoms are simple propositions, connectives have meaning in their truth tables, compound formulas get their meaning from combining these.

Remark: To parse a compound formula, it is sufficient to parse it and determine all sub-formulas and their truth values.

Definition. Truth Valuation: A truth valuation is a function $t : Atom(\mathcal{L}^p) \to \{0, 1\}$ which consumes an atom and returns its truth value. The truth table is a list of the truth value of a formula (and possibly sub-formulas) under all possible truth valuations. A single row of the truth table corresponds to a single permutation of truth valuations.

Definition. Evaluation of Formulas: Let t be a truth valuation and $A \in Form(\mathcal{L}^p)$. We define the value of A with respect to t to be

1. If $A = p \in Atom(\mathcal{L}^p)$, then $A^t = t(p)$ (the value assigned to p by t).
2. $(\neg B)^t = 1 - B^t$.
3. $(B \wedge C)^t = BC$.

4. $(B \vee C)^t = \max\{B, C\}$.
5. $(B \implies C)^t = ((\neg B) \vee C)^t$.
6. $(B \iff C)^t = ((B \wedge C) \wedge (\neg(B \vee C)))^t$.

Definition. Satisfied Formulas: We say a truth valuation t satisfies a formula $A \in Form(\mathcal{L}^p)$ if $A^t = 1$. Where Σ is a set of formulas, we defined

$$\Sigma^t = \begin{cases} 1 & \text{Each formula } B \in \Sigma \text{ has } B^t = 1 \\ 0 & \text{Otherwise} \end{cases}$$

Definition. Satisfiability: A set of formulas $\Sigma \subseteq Form(\mathcal{L}^p)$ is satisfiable if and only if there exists a truth valuation t such that $\Sigma^t = 1$. Otherwise, if for all truth valuations $\Sigma^t = 0$, Σ is said to be unsatisfiable. Further, if $\Sigma^t = 1$ then t is said to satisfy Σ and Σ is said to be satisfied by or under t.

Remark: If $\Sigma^t = 1$ then $\forall B \in \Sigma, B^t = 1$, if $\Sigma^t = 0$ then $\exists B \in \Sigma, B^t = 0$.

Definition. Tautology: A formula A is a tautology if for all possible truth valuations $A^t = 1$.

Definition. Contradiction: A formula A is a contradiction if for all possible truth valuations $A^t = 0$.

Definition. Contingent: A formula A is said to be contingent if it is neither a tautology nor a contradiction.

Proposition. Law of the Excluded Middle: $p \vee \neg p$ is a tautology. Also called *tertium non datur.*

Theorem: Let A be a tautology and let $p_1, \ldots, p_n$ be the propositional symbols of A. Suppose $B_1, \ldots, B_n \in Form(\mathcal{L}^p)$ then the formula obtained by replacing p_1 by B_1, ..., p_n by B_n is a tautology.

Proposition. Law of Contradiction: $\neg(p \wedge \neg p)$ is a tautology or $(p \wedge \neg p)$ is a contradiction.

Proposition. Plato's Essential Laws of Thought:

1. Law of identity: "whatever is, is." $p = p$
2. Law of contradiction: "nothing can bot be and not be." $\neg(p \wedge \neg p)$
3. Law of excluded middle: "Everything must either be, or not be." $(p \vee \neg p)$

Definition. Tautological Consequence: Suppose $\Sigma \subseteq Form(\mathcal{L}^p)$ and $A \in Form(\mathcal{L}^p)$. A is said to be a tautological consequence of Σ if for every truth valuation t such that $\Sigma^t = 1$ then $A^t = 1$. We also say Σ tautologically implies A. Symbolically this is $\Sigma \models A$.

Note: $\models$ is not a symbol of our language hence $\Sigma \models A \notin \mathcal{L}^p$. Rather, $\Sigma \models A$ is a statement in the metalanguage about Σ and A. $\Sigma \models A$ can also be expressed as the truth of formulas in Σ are sufficient to determine A is true.

Remark: A special tautological consequence is $\emptyset \models A$. Since $\emptyset^t$ is vacuously true for all truth valuations, $\emptyset \models A$ if and only if A is a tautology.

Proposition: Let $\Sigma = \{A_1, \dots, A_n\} \subseteq Form(\mathcal{L}^p)$ be a set of formulas (premises) and let $C \in Form(\mathcal{L}^p)$ be a formula (conclusion). The following are equivalent

1. The proposition with premises $A_1, \dots, A_n$ and C is valid.
2. $(A_1 \wedge \cdots \wedge A_n) \to C$ is a tautology.
3. $(A_1 \wedge \cdots \wedge A_n \wedge \neg C)$ is a contradiction or unsatisfiable.
4. C is a tautological consequence of Σ, i.e. $\Sigma \models C$.

Definition. Truth of Conclusion: Given an argument with premises $A_1, \dots, A_n$ and conclusion C, the conclusion C is said to be true if $\{A_1, \dots, A_n\} \models C$ and $A_1, \dots, A_n$ are all true.

Definition. Tautological Equivalence: For $A, B \in Form(\mathcal{L}^p)$, we say A and B are tautologically equivalent and write $A \mathrel{\text{⫤}\!\!\models} B$ when $A \models B$ and $B \models A$. In this case, for all truth valuations t, $A^t = B^t$.

Remark: Tautological and connective implication have different meanings. In particular, $A \implies B$ and $A \iff B$ are formulas, where as $A \models B$ if and only if $A \implies B$ is a tautology and $A \mathrel{\text{⫤}\!\!\models} B$ if and only if $A \iff B$ is a tautology.

Remark: One of the easiest ways to prove the tautological consequence $\Sigma \models A$ is to prove using a truth table that $A_1 \wedge \cdots \wedge A_n \implies A$ for $A_1, \dots, A_n \in \Sigma$.

Example: A simple example is $\{(p \implies q), (q \implies r)\} \models (p \implies r)$.

Note: A formula with n propositional symbols and m connectives has a truth table of 2^n rows and fewer than $n + m$ columns.

Remark. Proof by Contradiction: To prove $\Sigma \models C$, we can prove $\Sigma \not\models C$ is a contradiction.

Remark: To prove $\Sigma \not\models A$, we must construct a counterexample. A truth valuation t such that $\Sigma^t = 1$ and $A^t = 0$.

Theorem. De Morgan's Law: Using a truth table we can prove $\neg(p \wedge q) \mathrel{\text{⫤}\!\!\models} (\neg p \vee \neg q)$ and $\neg(p \vee q) \mathrel{\text{⫤}\!\!\models} (\neg p \wedge \neg q)$.

Theorem. Contrapositive: Using a truth table we can prove $(p \implies q) \mathrel{\text{⫤}\!\!\models} (\neg q \implies \neg p)$, these are called contrapositives of each other.

Remark. Converse: The converse of an implication $P \implies Q$ is $Q \implies P$. It is not true that $(P \implies Q) \mathrel{\text{⫤}\!\!\models} (Q \implies P)$ unless $P \mathrel{\text{⫤}\!\!\models} Q$.

Lemma: If $A \mathrel{\text{⫤}\!\!\models} A'$ and $B \mathrel{\text{⫤}\!\!\models} B'$ then

1. $\neg A \mathrel{\text{⫤}\!\!\models} \neg A'$.

2. $A \land B \models\!\!\!\!\!= A' \land B'$.
3. $A \lor B \models\!\!\!\!\!= A' \lor B'$.
4. $A \implies B \models\!\!\!\!\!= A' \implies B'$.
5. $A \iff B \models\!\!\!\!\!= A' \iff B$.

Theorem. Replaceability of Tautologically Equivalent Formulas: Let A be a formula which contains a sub-formula B. Suppose $B \models\!\!\!\!\!= C$ and let A' be the formula obtained by replacing any number of occurrences of B with C. Then $A' \models\!\!\!\!\!= A$. (Proof by structural induction).

Theorem. Duality: Suppose A is a formula composed only of atoms and the connectives $\neg, \lor, \land$, respecting their formation rules. Let $\Delta(A)$ be the formula obtained by simultaneously replacing all $\land$ with $\lor$, all $\lor$ with $\land$, and each atom with its negation. Then $\neg A \models\!\!\!\!\!= \Delta(A)$.

Remark. Fuzzy Logic: Fuzzy logic is a system using truth values in $[0, 1]$. True still is 1, false still is 0, all other values are said to be partially true. We denote $And(x, y) = \min\{x, y\}$, $Or(x, y) = \max\{x, y\}$, $Not(x) = 1 - x$. These definitions coincide with the definitions of classical logic. However, the law of excluded middle and the law of contradiction do not hold.

2.2 Propositional Calculus

Remark: Notice that some of the formulas used have 1 and/or 0, however these are not actually formulas or expression in $\mathcal{L}^p$ since $1, 0 \notin \mathcal{L}^p$. Rather they denote the values of a formula.

Law	Dual	Name
$A \lor \neg A \models\!\!\!\!\!= 1$		Excluded middle law
$A \land \neg A \models\!\!\!\!\!= 0$		Contradiction law
$A \lor 0 \models\!\!\!\!\!= A$	$A \land 1 \models\!\!\!\!\!= A$	Identity law
$A \lor 1 \models\!\!\!\!\!= 1$	$A \land 0 \models\!\!\!\!\!= 0$	Domination laws
$A \lor A \models\!\!\!\!\!= A$	$A \land A \models\!\!\!\!\!= A$	Idempotent laws
$\neg(\neg A) \models\!\!\!\!\!= A$		Double-negation law
$A \lor B \models\!\!\!\!\!= B \lor A$	$A \land B \models\!\!\!\!\!= B \land A$	Commutativity laws
$(A \lor B) \lor C \models\!\!\!\!\!= A \lor (B \lor C)$	$(A \land B) \land C \models\!\!\!\!\!= A \land (B \land C)$	Associativity laws
$A \lor (B \land C) \models\!\!\!\!\!= (A \lor B) \land (A \lor C)$	$A \land (B \lor C) \models\!\!\!\!\!= (A \land B) \lor (A \land C)$	Distributivity laws
$\neg(A \land B) \models\!\!\!\!\!= \neg A \lor \neg B$	$\neg(A \lor B) \models\!\!\!\!\!= \neg A \land \neg B$	De Morgan's laws
$A \iff B \models\!\!\!\!\!= (A \implies B) \land (B \implies A)$	$A \iff B \models\!\!\!\!\!= (A \land B) \lor (\neg A \land \neg B)$	Iff Definition
$A \implies B \models\!\!\!\!\!= \neg A \lor B$	$A \iff B \models\!\!\!\!\!= (\neg A \lor B) \land (\neg B \lor A)$	Connective removal
$A \lor (A \land B) \models\!\!\!\!\!= A$	$A \land (A \lor B) \models\!\!\!\!\!= A$	Absorption laws
$(A \land B) \lor (\neg A \land B) \models\!\!\!\!\!= B$	$(A \lor B) \land (\neg A \lor B) \models\!\!\!\!\!= B$	Redundancy law

Definition. Literal: A formula is called a literal if it is of the form p or $\neg p$ for some propositional symbol, p and $\neg p$ are called complementary literals.

Definition. Clause: A disjunction of literals is called a (disjunctive) clause. A conjuction of literals is called a (conjunctive) clause.

Example: $(p \vee q \vee \neg r)$ is a disjunctive clause. $(\neg p \wedge s \wedge \neg q \wedge r)$ is a conjunctive clause.

Definition. Normal Form: A disjunction of conjunctive clauses is said to be in disjunctive normal form (DNF). A conjunction of disjunctive clauses is said to be in conjuctive normal form (CNF).

Example: $(p \wedge q) \vee (p \wedge \neg q) \vee p$ and $\neg p \vee q$ are in disjunctive normal form. $\neg(p \wedge q) \vee r$ is not in disjunctive normal form. $p \vee (r \wedge (p \vee q))$ is not in disjunctive normal form.

Remark: Atoms are literals, disjunctive clauses, conjunctive clauses, and in both DNF and CNF.

Remark. Algorithm for CNF:

1. Eliminate if and only if and implications using removal of conjunctive laws.
2. Use De Morgan's law and double-negation to obtain an equivalent formula where each $\neg$ has a scope of a single atom.
3. If A is literal, then $CNF(A) = A$.
4. If A is $B \wedge C$, then $CNF(A) = CNF(B) \wedge CNF(C)$.
5. If A is $B \vee C$, then suppose $CNF(B) = B_1 \wedge \cdots \wedge B_n$ and $CNF(C) = C_1 \wedge \cdots \wedge C_n$. Then $CNF(C) = \wedge_{1 \leq i \leq n, 1 \leq j \leq m}(B_i \vee C_j)$.

Notice the last step is similar to using distributivity to expand $(x_1 + \cdots + x_n)(y_1 + \cdots + y_n)$.

Theorem: Any formula $A \in Form(\mathcal{L}^p)$ is tautologically equivalent to some formula in disjunctive normal form.

Proof. If A is a contradiction, then A is tautologically equivalent to $p \wedge \neg p$. Otherwise, suppose A has propositional symbols $p_1, \ldots, p_n$. For each truth valuation t, we can generate a conjunctive clause C such that $C^t = A^t$. Indeed, if $A^t = 1$, then $(\wedge_{1 \leq i \leq n, p_i^t = 1} p_i) \wedge_{1 \leq i \leq n, p_i^t = 0} \neg p_i)$. If $A^t = 0$, then we can safely ignore the case as all other clauses will yield 0 for these. $\square$

Theorem: Any formula $A \in Form(\mathcal{L}^p)$ is tautologically equivalent to some formula in conjunctive normal form.

Proof. Use duality theorem. $\square$

Week 3 Adequacy and Formal Deduction

3.1 Adequate Sets of Connectives

Definition. Reducability: A connective $\star$ which can be written as the formula of other connectives is said to be reducible to those formulas.

Example: $\implies$ reduces to $\neg$ and $\vee$ as $A \implies B \models\mathrel{\mkern-6mu}\dashv \neg A \vee B$, Similarly, $\vee$ reduces to $\neg$ and $\implies$.

Definition. n-ary Connective: We use letters such as f to denote connectives. For instance, $f(A_1, \ldots, A_n)$ is the formula formed by the n-ary connective f connecting $A_1, \ldots, A_n$. A connective is defined its truth table and we say two connective are the same if and only if they have the same truth tables.

Remark: It is obvious to prove by truth tables that there are 4 unique unary connectives. Similarly, we can show there are 16 binary connectives. In general, for $n \in \mathbb{N}$, an n-ary connective's truth table contains 2^n rows and each entry in each row can be mapped to one of two outcomes (either $x \mapsto 1$ or $x \mapsto 0$). Hence for each row, there are 2^{2^n} mappings and hence this many n-ary connectives.

Definition. Adequacy: Where S is a set of connectives, we say S is adequate if and only if for every n-ary connective f with $n \geq 1$, there is a formula A_S written using only connectives in S such that $f(p_1, \ldots, p_n) \models\mathrel{\mkern-6mu}\dashv A_S$, where p_i are proposition symbols for $1 \leq i \leq n$.

Example: The set of standard connectives $\{\neg, \wedge, \vee, \implies, \iff\}$ for instance is adequate. Notice an adequate set is one which can describe every truth table or every n-ary connective.

Theorem: The set $S_0 = \{\neg, \wedge, \vee\}$ is an adequate set of connectives.

Proof. Let f be an arbitrary n-ary connective. Constructing the truth table for $f(p_1, \ldots, p_n)$, we use the theorem about the existence of disjunctive normal forms to obtain a formula A_{S_0} in DNF with $f(p_1, \ldots, p_n) \models\mathrel{\mkern-6mu}\dashv A_{S_0}$. Since A_{S_0} uses only connectives in $S_0 = \{\neg, \wedge, \vee\}$, S_0 is adequate. $\square$

Remark: Similarly, this implies that any set S of connectives can be shown to be adequate by showing that each connective in S_0 is definable in the connectives in S. This is since any formula can be written in S_0 and then rewritten into S using the replaceability theorem.

Corollary: $\{\neg, \wedge\}$, $\{\neg, \vee\}$, $\{\neg, \implies\}$ are adequate.

Proof. Notice $A \vee B \models\mathrel{\mkern-6mu}\dashv \neg(\neg A \wedge \neg B)$, hence the first set is adequate. Notice $A \wedge B \models\mathrel{\mkern-6mu}\dashv \neg(\neg A \vee \neg B)$ hence the second set is adequate. Notice $A \wedge B \models\mathrel{\mkern-6mu}\dashv \neg(A \implies \neg B)$ and $A \vee B \models\mathrel{\mkern-6mu}\dashv \neg A \implies B$. $\square$

Definition. Peirce Arrow: Also called NOR and denoted $\downarrow$, it is defined as

p	q	$p \downarrow q$
1	1	0
1	0	0
0	1	0
0	0	1

Notice $p \downarrow q \vDash\Dashv \neg(p \lor q)$.

Proposition. Peirce Arrow is Adequate: The set $\{\downarrow\}$ is adequate.

Proof. Notice $\neg p \vDash\Dashv p \downarrow p$, and $p \land q \vDash\Dashv (p \downarrow p) \downarrow (q \downarrow q)$, and $p \lor q \vDash\Dashv (p \downarrow q) \downarrow (p \downarrow q)$. □

Definition. Sheffer Stroke: Also called NAND and denoted $|$, it is defined as

p	q	$p \vert q$
1	1	0
1	0	1
0	1	1
0	0	0

Notice $p|q \vDash\Dashv \neg(p \land q)$.

Proposition. Sheffer Stroke is Adequate: The set $\{|\}$ is adequate.

Proof. Notice $\neg p \vDash\Dashv p|p$, and $p \land q \vDash\Dashv (p|q)|(p|q)$, and $p \lor q \vDash\Dashv (p|p)|(q|q)$. □

Remark. Proving Inadequacy: To prove S is inadequate, it suffices to show that one of the connectives in S_0 cannot be defined using the connectives in S.

Example: $\{\land\}$ is inadequate since $p \land p \vDash\Dashv p$, hence there is no way to define the unary connective $\neg$.

Definition. If-Then-Else Operator: Let τ be ternary connective with truth table

p	q	r	$\tau(p,q,r)$
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

Or $\tau(p,q,r) \vDash\Dashv (p \implies q) \land (\neg p \implies r)$

3.2 Formal Deduction

Remark. Sets as Sequences: For convenience, we write sets and unions thereof as sequences. For instance, if $\Sigma = \{A_1, A_2, \ldots\}$, we may write that Σ is $A_1, A_2, \ldots$. Or $\Sigma \cup \{A\}$ as Σ, A and $\Sigma \cup \Sigma'$ as Σ, Σ'.

Definition. Deducibility: We write $\Sigma \vdash A$ to mean A is formally deducible or provable from Σ.

Definition. Sequent and Proofs: A statement of the form $\Sigma_i \vdash A_i$ is called a sequents. A finite sequence of sequents is called a proof, and a proof is said to be valid if each of its sequents is created from previous ones according to the specified proof rules. A theorem is the last sequent to appear in the valid proof.

Definition. Eleven Rules of Natural Deduction: For every formula A, B, C and sets of formulas Σ, Σ':

1.	(Ref)	$A \vdash A$ is a theorem.	Reflexivity
2.	(+)	If $\Sigma \vdash A$ is a theorem, then $\Sigma, \Sigma' \vdash A$ is a theorem.	Addition of Premises
3.	$(\neg -)$	If $\Sigma, \neg A \vdash B$ and $\Sigma, \neg A \vdash \neg B$ are theorems, then $\Sigma \vdash A$ is a theorem.	$\neg$ elimination
4.	$(\rightarrow -)$	If $\Sigma \vdash A \implies B$ and $\Sigma \vdash A$ are theorems, then $\Sigma \vdash B$ is a theorem.	$\implies$ elimination
5.	$(\rightarrow +)$	If $\Sigma, A \vdash B$ is a theorem, then $\Sigma \vdash A \implies B$ is a theorem.	$\implies$ introduction
6.	$(\wedge -)$	If $\Sigma \vdash A \wedge B$ is a theorem, then $\Sigma \vdash A$ and $\Sigma \vdash B$ are theorems.	$\wedge$ elimination
7.	$(\wedge +)$	If $\Sigma \vdash A$ and $\Sigma \vdash B$ are theorems then $\Sigma \vdash A \wedge B$ is a theorem.	$\wedge$ introduction
8.	$(\vee -)$	If $\Sigma, A \vdash C$ and $\Sigma, B \vdash C$ are theorems then $\Sigma, A \vee B \vdash C$ is a theorem.	$\vee$ elimination
9.	$(\vee +)$	If $\Sigma \vdash A$ is a theorem, then $\Sigma \vdash A \vee B$ and $\Sigma \vdash B \vee A$ are theorems.	$\vee$ introduction
10.	$(\leftrightarrow -)$	If $\Sigma \vdash A \iff B$ and $\Sigma \vdash A$ are theorems, then $\Sigma \vdash B$ is a theorem. If $\Sigma \vdash A \iff B$ and $\Sigma \vdash B$ are theorems, then $\Sigma \vdash A$ is a theorem.	$\iff$ elimination.
11.	$(\leftrightarrow +)$	If $\Sigma, A \vdash B$ and $\Sigma, B \vdash A$ is a theorem, then $\Sigma \vdash A \iff B$ is a theorem	$\iff$ introduction.

Remark. Intuition: Reflexivity: anything you know you may deduce. Addition: additional information does not destroy knowledge. $\wedge$ elimination / introduction: to know each of two things is the same to know both of them. $\implies$ elimination / introduction: we may deduce something if and only if it is implied. $\vee$ elimination: proof by cases. $\neg$ elimination: proof by contradiction.

Theorem. Membership Rule: If $A \in \Sigma$, then $\Sigma \vdash A$.

Proof. Suppose $A \in \Sigma$, let $\Sigma' = \Sigma \setminus \{A\}$. By reflexivity $A \vdash A$, by addition of premises $A, \Sigma' \vdash A$ hence $\Sigma \vdash A$. □

Theorem. Transitivity of Implication: $A \implies B, B \implies C \vdash A \implies C$.

Proof. By the membership rule, $A \implies B, B \implies C, A \vdash A \implies B$, (1) and $A \implies B, B \implies C, A \vdash B \implies C$, (2) and $A \implies B, B \implies C, A \vdash A$ (3). By implication elimination: on (1) and (2) $A \implies B, B \implies C, A \vdash B$ (4) and on (3) and (4) $A \implies B, B \implies C, A \vdash C$ (5). By implication introduction on (5) $A \implies B, B \implies C \vdash A \implies C$. □

Definition. Schema: A schema is a rule which once proved can be used in proofs (while maintaining validity). For instance, the eleven rules of natural deduction, the membership rule, and the transitivity of implication are all schema. A lemma is any minor schema used in larger theorems.

Definition. Scheme of Formal Deducibility: Also called a theorem, it is a demonstrated $\Sigma \vdash A$ (for which we have a formal proof). These rules are purely syntactic and can be checked mechanically.

Definition. Formal Deducibility: A formula A is formally deducible from Σ, written $\Sigma \vdash A$ if and only if $\Sigma \vdash A$ is generated by finitely many applications of the rules of a formal system. I.e. there is a sequence $\Sigma_1 \vdash A_1, \ldots, \Sigma_n \vdash A_n$ such that each sequent is generated by a formal deduction of previous terms and $\Sigma_n \vdash A_n$ is $\Sigma \vdash A$.

Example: To say $\Sigma_k \vdash A_k$ is generated by a rule of formal deduction is to say that the subsequence $\Sigma_1 \vdash A_1, \ldots, \Sigma_{k-1} \vdash A_{k-1}$ which precedes $\Sigma_k \vdash A_k$ has lines of the form $\Sigma_i, \neg A_i \vdash B$ and $\Sigma_j, \neg A_j \vdash \neg B$.

Definition. Formal Proof: The sequence $\Sigma_1 \vdash A_1, \ldots, \Sigma_n \vdash A_n$ is called a formal proof of $\Sigma_n \vdash A_n$. In this case we say A is formally deducible from Σ. Notice the definition of formal deducibility is is an inductive one stemming from the basis $A \vdash A$.

Week 4 Formal Deduction and Soundness

4.1 More Formal Deduction

Remark. Difference between $\models$ and $\vdash$: $A \models B$ if $A \implies B$ is a tautology. A**Proposition. B:** if $\emptyset \vdash A \implies B$

Theorem. Reductio as Absurdum: Shorthand $(\neg+)$, if $\Sigma, A \vdash B$ and $\Sigma, A \vdash \neg B$ are theorems then $\Sigma \vdash \neg A$

$$\begin{aligned}
&\Sigma, A \vdash B && \text{Given} && (1)\\
&\Sigma, A \vdash \neg B && \text{Given} && (2)\\
&\Sigma \vdash A \implies B && \rightarrow + \text{ on } 1 && (3)\\
&\Sigma \vdash A \implies \neg B && \rightarrow + \text{ on } 2 && (4)\\
&\Sigma, \neg\neg A \vdash A \implies B && + \text{ on } 3 && (5)\\
&\Sigma, \neg\neg A \vdash A \implies \neg B && + \text{ on } 4 && (6)\\
&\neg\neg A, \neg A \vdash \neg A && \in && (7)\\
&\neg\neg A, \neg A \vdash \neg\neg A && \in && (8)\\
&\neg\neg A \vdash A && \neg- \text{ on 7 and 8} && (9)\\
&\Sigma, \neg\neg A \vdash A && + \text{ on } 9 && (10)\\
&\Sigma, \neg\neg A \vdash B && \rightarrow - \text{ using 9 on 5} && (11)\\
&\Sigma, \neg\neg A \vdash \neg B && \rightarrow - \text{ using 9 on 6} && (12)\\
&\Sigma \vdash B && \neg- \text{ on 11 and 12} && (13)
\end{aligned}$$

Note the proof of $\neg\neg A \vdash A$ is sometimes denoted $\neg\neg-$.

Theorem. Proof by Contrapositive: For all A, B we have $A \to B \vdash \neg B \to \neg A$.

$$
\begin{array}{rlr}
A \to B, \neg B, A \vdash A \to B & \in & (1) \\
A \to B, \neg B, A \vdash A & \in & (2) \\
A \to B, \neg B, A \vdash B & \to - \text{ using 2 on 1} & (3) \\
A \to B, \neg B, A \vdash \neg B & \in & (4) \\
A \to B, \neg B \vdash \neg A & \neg + \text{ on 3 and 4} & (5) \\
A \to B \vdash \neg B \to A & \to + \text{ on 5} & (6)
\end{array}
$$

Remark: We say A is formally provable from nothing if $\emptyset \vdash A$. In particular $\emptyset \vdash A$ if and only if $\Sigma \vdash A$ for all Σ. Examples of such formulae are $\emptyset \vdash \neg(A \wedge \neg A)$ and $\emptyset \vdash A \vee \neg A$.

Theorem. Transitivity of Deducibility: Denoted (Tr). Let $\Sigma \subseteq Form(\mathcal{L}^p)$ and $A, A_1, \dots, A_n \in Form(\mathcal{L}^p)$. If $\Sigma \vdash A_i$ for all $1 \leq i \leq n$ and $A_1, \dots, A_n \vdash A$ then $\Sigma \vdash A$. Note $n < \infty$.

Proof.

$$
\begin{array}{rlr}
A_1, \dots, A_n \vdash A & \text{Given} & (1) \\
A_1, \dots, A_{n-1} \vdash A_n \implies A & \to + \text{ on } 1 & (2) \\
A_1, \dots, A_{n-2} \vdash A_{n-1} \implies (A_n \implies A) & \to + \text{ on } 2 & (3) \\
\vdots & & \\
\emptyset \vdash A_1 \implies (\cdots (A_n \implies A)) & \to + \text{ on } n & (n+1) \\
\Sigma \vdash A_1 \implies (\cdots (A_n \implies A)) & + \text{ on } n+1 & (n+2) \\
\Sigma \vdash A_1 & \text{Given} & (n+3) \\
\Sigma \vdash A_2 \implies (\cdots (A_n \implies A)) & \to - \text{ using } n+3 \text{ on } n+2 & (n+4) \\
\vdots & & \\
\Sigma \vdash A_n \implies A & \to - \text{ using } 3n \text{ on } 3n-1 & (3n+1) \\
\Sigma \vdash A_n & \text{Given} & (3n+2) \\
\Sigma \vdash A & \to - \text{ using } 3n+2 \text{ on } 3n+1 & (3n+3)
\end{array}
$$

□

Remark: We may write $\Sigma \vdash A_1, \dots, A_n$ to mean $\Sigma \vdash A_1$ and ... and $\Sigma \vdash A_n$. Using this notation, (Tr) can be written as "if $\Sigma \vdash A_1, \dots, A_n$ and $A_1, \dots, A_n \vdash A$ then $\Sigma \vdash A$."

Notation: We may write $A \dashv B$ to mean $B \vdash A$, i.e. the reverse of $\vdash$. We may also write $A \vdash\dashv B$ to means $A \vdash B$ and $B \vdash A$.

Definition. Syntactically Equivalent: We say A and B are syntactically equivalent if $A \vdash\dashv B$.

Lemma: If $A \vdash\dashv A'$ and $B \vdash\dashv B'$ then

1. $\neg A \vdash\dashv \neg A'$
2. $A \wedge B \vdash\dashv A' \wedge B'$
3. $A \vee B \vdash\dashv A' \vee B'$
4. $A \implies B \vdash\dashv A' \implies B'$
5. $A \iff B \vdash\dashv A' \iff B'$

Theorem: Suppose $B \vdash\dashv C$. For any A let A' be constructed from A by replacing any number occurrences of B by C. Then $A \vdash\dashv A'$.

Definition. Formal System: A formal system is a set of rules, such as those eleven for the formal system Natural Deduction. A formal system is one such that every statement one can prove is actually correct (soundness) and one should be able to prove within the system every correct statement (completeness).

Theorem: The formal system of natural deduction is sound for propositional logic. That is for every Σ and A, if $\Sigma \vdash A$ then $\Sigma \models A$. Natural deduction is complete for propositional logic. That is for every Σ and A, if $\Sigma \models A$ then $\Sigma \vdash A$.

Remark: Natural deduction cannot prove the invalidity of a argument. For instance, we cannot prove the fallacy $p \implies q, \neg p \vdash \neg q$ is invalid. We state instead that each argument with no proof of validity is invalid.

4.2 Soundness

Definition. Inconsistent: A set of formulas Σ is inconsistent if there is a formula A such that $\Sigma \vdash A$ and $\Sigma \vdash \neg A$.

Lemma: A set Σ is inconsistent if and only if for every formula B, we have $\Sigma \vdash B$

Proof. ($\impliedby$) Let B be a formula. Then by $(+)$ $\Sigma, B \vdash A$ and $\Sigma, B \vdash \neg A$ thus by $(\neg -)$ $\Sigma \vdash B$. □

Definition. Consistent: A set Σ is consistent if it is not inconsistent. I.e. there is no A such that $\Sigma \vdash A$ and $\Sigma \vdash \neg A$.

Lemma: A set Σ is consistent if and only if there is a formula B such that $\Sigma \nvdash B$.

Lemma: Let Σ be a set of formulas and A be a formula. $\Sigma \vdash A$ if and only if $\Sigma, \neg A$ is inconsistent or equivalently $\Sigma \nvdash A$ if and only if $\Sigma, \neg A$ is consistent.

Proof. ($\implies$) $\Sigma \vdash A$, $(+)$ $\Sigma, \neg A \vdash A$, $(\in)$ $\Sigma, \neg A \vdash \neg A$.

($\impliedby$) Since $\Sigma, \neg A$ is inconsistent $\Sigma, \neg A \vdash A$, $(\in)$ $\Sigma, \neg A \vdash \neg A$, $(\neg -)$ $\Sigma \vdash A$. □

Definition. Maximal: A consistent set Σ is maximal if for every formula A either $A \in \Sigma$ or $\neg A \in \Sigma$ (not both since Σ is consistent).

Lemma: If Σ is maximal, then for every A, $\Sigma \vdash A$ if and only if $A \in \Sigma$.

Proof. Let A be a formula. ($\implies$) Suppose $\Sigma \vdash A$. Since $\Sigma \vdash A$, $\Sigma \nvdash \neg A$. Hence since Σ is maximal and thus consistent, $\neg A \notin \Sigma$ and thus $A \in \Sigma$.

($\impliedby$) Suppose $A \in \Sigma$. By ($\in$), $\Sigma \vdash A$. □

Remark: A formal deduction proof can be defined inductively. Our base case is $\Sigma \vdash C$ for some $C \in \Sigma$ by ($\in$). The inductive step to apply an inference rule.

Theorem. Soundness of Propositional Formal Deduction: For all Σ and C, if $\Sigma \vdash C$ then $\Sigma \models C$.

Proof. Base case: must be (Ref), i.e. $\Sigma = \{C\}$. Now $\{C\} \models C$ as required.

Assume by the inductive hypothesis that $\Sigma \vdash C$ and $\Sigma \models C$ for a sequent Σ. We do case analysis by our possible rules.

1. (Ref) See base case.

2. (+) The current line is $\Sigma, \Sigma' \vdash A$ where $\Sigma \vdash A$ previously. Now $\Sigma \models A$ so for all valuations t, if $\Sigma^t = 1$ then $A^t = 1$. Notice if $(\Sigma, \Sigma')^t = 1$ then $\Sigma^t = 1$ hence $A^t = 1$. Thus $\Sigma, \Sigma' \vdash A$ and $\Sigma, \Sigma' \models A$.

3. ($\neg-$) The current line is $\Sigma \vdash A$ where $\Sigma, \neg A \vdash B$ and $\Sigma, \neg A \vdash \neg B$ previously. By hypothesis $\Sigma, \neg A \models B$ and $\Sigma, \neg A \models \neg B$. This requires for all truth valuations t, $(\Sigma, \neg A)$. I.e. if $\Sigma^t = 1$ then $(\neg A)^t = 0$ hence $\Sigma \models A$.

4. ($\to -$) The current line is $\Sigma \vdash B$ where $\Sigma \vdash A$ and $\Sigma \vdash A \to B$ previously. So $\Sigma \models A \to B$ and $\Sigma \models A$. Thus whenever $\Sigma^t = 1$ then $A^t = 1$ and $(A \to B)^t = 1$. However, since $(A \to B)^t = 1$ then whenever $A^t = 1$ then $B^t = 1$, thus $B^t = 1$ whenever $\Sigma^t = 1$, i.e. $\Sigma \models B$.

5. ($\to +$) The current line is $\Sigma \vdash A \to B$ where $\Sigma, A \vdash B$ previously. So $\Sigma, A \models B$, so whenever $\Sigma^t = 1$ and $A^t = 1$ then $B^t = 1$. Thus whenever $\Sigma^t = 1$, then if $A^t = 1$ also, then $B^t = 1$, hence whenever $\Sigma^t = 1$ then $(A \to B)^t = 1$, i.e. $\Sigma \models A \to B$.

6. ($\wedge-$) The current line is $\Sigma \vdash A$ where $\Sigma \vdash A \wedge B$ or $\Sigma \vdash B \wedge A$ previously. Thus either $\Sigma \models A \wedge B$, hence if $\Sigma^t = 1$ then $(A \wedge B)^t = 1$ thus $A^t = 1$ hence $\Sigma \models A$.

7. ($\wedge+$) The current line is $\Sigma \vdash A \wedge B$ where $\Sigma \vdash A$ and $\Sigma \vdash B$ previously. So whenever $\Sigma^t = 1$, $A^t = 1$ and $B^t = 1$. Hence $\Sigma \models A \wedge B$.

8. ($\vee-$) The current line is $\Sigma, A \vee B \vdash C$ where $\Sigma, A \vdash C$ and $\Sigma, B \vdash C$. So whenever $\Sigma^t = 1$ and $A^t = 1$ then $C^t = 1$, however, whenever $\Sigma^t = 1$ and $B^t = 1$ then $C^t = 1$. Thus whenever $A^t = 1$ or $B^t = 1$ then $C^t = 1$, i.e. $\Sigma, A \vee B \models C$.

9. ($\vee+$) The current line is $\Sigma \vdash A \vee B$ where $\Sigma \vdash A$ or $\Sigma \vdash B$ previously. So $\Sigma \models A$ or $\Sigma \models B$, regardless $\Sigma \models A \vee B$.
10. ($\leftrightarrow -$) The current line is $\Sigma \vdash A$ where $\Sigma \vdash B$ and $\Sigma \vdash A \leftrightarrow B$ or $\Sigma \vdash B \leftrightarrow A$ previously. So whenever $\Sigma^t = 1$ then $B^t = 1$ and $A \leftrightarrow B^t = 1$ so $A^t = 1$ therefore $\Sigma \models A$.
11. ($\leftrightarrow +$) The current line is $\Sigma \vdash A \leftrightarrow B$ where $\Sigma \vdash A$ and $\Sigma \vdash B$ previously. So whenever $\Sigma^t = 1$, then $A^t = 1$ and $B^t = 1$ therefore $(A \leftrightarrow B)^t = 1$ so $\Sigma \models (A \leftrightarrow B)$.

□

Example. Application of Soundness: Prove that $A \to B \nvdash B \to A$. We know the valuation such that $A^t = 0$ and $B^t = 1$ has $(A \to B)^t = 1$ but $(B \to A)^t = 0$ so $A \to B \not\models B \to A$ and by soundness no such proof can exist.

Theorem. Completeness of Propositional Formal Deduction: If $\Sigma \models A$ then $\Sigma \vdash A$.

Proof. Recall that $\Sigma \nvdash A$ if and only if $\Sigma, \neg A$ is consistent and $\Sigma \not\models A$ if and only if $\Sigma, \neg A$ is satisfiable. Thus we prove that every consistent set is satisfiable.

Let Σ be a consistent set only using $\neg, \wedge, \vee, \to$. Since $\mathcal{L}^p$ is countable there is a sequence of all formulas in $Form(\mathcal{L}^p)$ $A_1, A_2, \ldots$. Let $\Sigma_0 = \Sigma$ and for $i \geq 0$

$$\Sigma_{i+1} = \begin{cases} \Sigma_i \cup \{A_i\} & \text{if } \Sigma_i \cup \{A_i\} \text{ is consistent} \\ \Sigma_i & \text{otherwise} \end{cases}$$

Since each Σ_i is consistent, $M = \bigcup_{i=0}^{\infty} \Sigma_i$ is consistent. Also notice M is maximal since if $A \notin M$ then $M \cup \{A\}$ is inconsistent and so $M \cup \{\neg A\}$ is consistent meaning $\neg A \in M$. Notice that if M is satisfiable, then Σ is satisfiable, and so each our proof is complete.

Let t be the truth valuation such that for each proposition symbol p, if $p \in M$ then $p^t = 1$, if $\neg p \in M$ then $p^t = 0$. Since M is maximal, each symbol has exactly one value.

We prove a lemma. For every formula C, $C^t = 1$ if and only if $C = 1$.

Notice that for every A and B if $A \in M$ and $A \to B \in M$ then $B \in M$ since M is maximal. Also notice that for every $B \in M$ and A, $M \vdash B$ thus $M, A \vdash B$ meaning $M \vdash A \to B$ or $A \to B \in M$. Lastly, for every $A \notin M$ and B, $\neg A \in M$ and $M \vdash \neg A$ but also $M, A, \neg A, \vdash B$ so $M \vdash A \to B$.

We now perform induction. Our base case is where C is a proposition symbol, $C^t = 1$ if and only if $C \in M$. We assume for our inductive hypothesis (IH) that $A^t = 1$ if and only if $A \in M$.

1. $C = \neg A$. If $C^t = 1$, then $A^t = 0$ so by our IH $A \notin M$ and thus $\neg A \in M$ as desired. If $C \in M$, then $\neg A \in M$ and thus $A \notin M$, by our IH $A^t = 0$ or $\neg A^t = 1$.

2. $C = A \land B$. If $C^t = 1$ then by our IH $A \in M$ and $B \in M$. So by $(\land+)$ $M \vdash A \land B$ and thus $A \land B \in M$. If $C \in M$ then by $(\land-)$ $A \in M$ and $B \in M$ and so by our IH $A^t = 1$ and $B^t = 1$ meaning $(A \land B)^t = 1$.

3. $C = A \lor B$. If $C^t = 1$ then $A^t = 1$ or $B^t = 1$. If $A^t = 1$ then by our IH $M \vdash A$ so by $(\lor+)$ $M \vdash A \lor B$ and thus $(A \lor B) \in M$. If $B^t = 1$ the proof is similar. If $C^t \in M$ then suppose $A^t = 0$ and $B^t = 0$. By IH $A \notin M$ and $B \notin M$ so $\neg A \in M$ and $\neg B \in M$. So by $(\land+)$ $M \vdash \neg A \land \neg B$ thus $\neg(A \lor B) \in M$, however this is a contradiction since M is consistent.

4. $C = A \to B$. If $C^t = 1$ then $B^t = 1$ or $A^t = 0$. If $B^t = 1$ then by IH $B \in M$ and so by a remark above $A \to B \in M$. If $A^t = 1$ then by IH $A \notin M$ and so by a remark above $A \to B \in M$. If $C \in M$ and $A \in M$, then by IH $A^t = 1$ and so by a remark above $B \in M$, thus $B^t = 1$ and so $(A \to B)^t = 1$. If $A \notin M$ then $A^t = 0$ and so $(A \to B)^t = 1$.

□

Theorem. Equivalent Definition of Soundness: The following are equivalent: (1) if $\Sigma \vdash A$ then $\Sigma \models A$ and (2) if Σ is satisfiable then Σ is consistent.

Proof. ($\implies$) Suppose Σ being satisfiable implies Σ is consistent. The contrapositive is that $\Sigma, \neg A$ being inconsistent implies $\Sigma, \neg A$ is unsatisfiable. $\Sigma, \neg A$ being inconsistent means $\Sigma \vdash A$. $\Sigma, \neg A$ being unsatisfiable means $\Sigma \models A$.

($\impliedby$) Apply the above steps in reverse. □

Unit 2 First-Order Logic

Week 5 Resolution, First Order Logic

5.1 Resolution for Propositional Logic

Definition. System of Resolution: Resolution is a system of formal deduction with a single rule, the rule of resolution.

Definition. Resolution Rule: Resolution is the formal deduction rule $C \lor p, D \lor \neg p \vdash_r C \lor D$ where C and D are disjunctive clauses and p is a literal. We say $C \lor p$ and $D \lor \neg p$ are parent clauses resolving over p. $C \lor D$ is called the resolvent. The resolvent of p and $\neg p$ is the empty clauses $\{\}$, this is not satisfiable.

Definition. Resolution: Also called derivation, the resolution of a set of a clauses S is a finite sequence of clauses such that each is either in S or resolved from previous clauses in the sequence by resolution.

Definition. Resolvable: Two clauses can be resolved if and only if they contain two complementary literals, say p and $\neg p$, in this case we say that we resolved over p or the resolution is on p.

Remark: To prove C from $A_1, \ldots, A_n$, show that $\{A_1, \ldots, A_n, \neg C\}$ is not satisfiable or by the resolution procedure that $A_1, \ldots, A_n, \neg C \vdash_r \{\}$.

Remark. Resolution Procedure: Given a set of disjunctive clauses $S = \{D_1, \ldots, D_m\}$ choose two clauses, one with p and one with $\neg p$ for some symbol p and resolve and the resolvent D. If $D = \{\}$ then output empty clause, otherwise add D to S. Repeat until you produce the empty clause.

Theorem. Soundness of Resolution: The resolvent it tautologically implied by its parent clauses, i.e. resolution is a sound rule of formal deduction.

Proof. Let p be a proposition symbol and let A and B be clauses. We have $p \vee A, \neg p \vee B \vdash_r A \vee B$ and wish to show $p \vee A, \neg p \vee B \models A \vee B$. Consider a truth valuation t such that $(p \vee A)^t = (\neg p \vee B)^t = 1$. Notice if $p^t = 0$ then $A^t = 1$, otherwise if $p^t = 1$ then $B^t = 1$. Regardless $(A \vee B)^t = 1$. However, if both A and B are both empty then we have $p, \neg p \vdash_r \{\}$ and $p, \neg p \models \{\}$ since it is a contradiction. □

Definition. Set-of-Support: Partition the clauses into two sets, the set of support and the auxillary set. The auxillary set is such that no formulae are contradictory, generally the initial set of premises is the auxillary set and the negation of the conclusion is set of support. Since contradiction cannot be derived from the auxillary set, we use at least on clause from the set of support. You then add the resolvent to the set of support. After each step, you perform each possible resolution with the clauses in the set of support.

Theorem: Resolution with the set-of-support strategy is complete.

Theorem. Pigeonhole Principle: One cannot put $n+1$ objects into n slots with distinct objects going into distinct slots.

Proof. For $1 \leq i \leq n+1$ and $1 \leq j \leq n$, let p_{ij} be the proposition symbol which is true if the ith pigeon goes into the jth slot. We construct clauses for each pigeon $1 \leq i \leq n+1$ going into some slot k for $1 \leq k \leq n$. That is $p_{i1} \vee p_{i2} \vee \cdots \vee p_{in}$. Distinct pigeons cannot go into the same slot, i.e. $p_{ik} \to \neg p_{jk} \models\mathrel{\mkern-3mu}\Dashv \neg p_{ik} \vee \neg p_{jk}$ for all $1 \leq i < j \leq n+1$ and $1 \leq k \leq n$. Any truth valuation which satisfies the above would contradict the pigeonhole principle, hence it is unsatisfiable (prove). □

Example. Pigeon hole principle for $n = 2$: Our disjunctions would be each pigeon in some slot: $p_{11} \vee p_{12}, p_{21} \vee p_{22}, p_{31} \vee p_{32}$. No two pigeons per slot. Slot 1: $\neg p_{11} \vee \neg p_{21}, \neg p_{11} \vee \neg p_{31}, \neg p_{21} \vee \neg p_{31}$. Slot 2: $\neg p_{12} \vee \neg p_{22}, \neg p_{12} \vee \neg p_{32}, \neg p_{22} \vee \neg p_{32}$.

Definition. David-Putnam Procedure: Abbreviated DPP, it is a procedure to formally prove using resolution. First write disjunctive clauses as sets of literals, i.e. $p \vee \neg q \vee r$ to $\{p, \neg q, r\}$. Let S be the set of clauses (as sets) and suppose it has propositional symbols $p_1, \ldots, p_n$. Let $S_1 = S$ and $i = 1$. Loop

1. Write disjunctive clauses as sets of literals, i.e. $p \vee \neg q \vee r$ to $\{p, \neg q, r\}$. Let S be the set of clauses (as sets) and suppose it has propositional symbols $p_1, \ldots, p_n$.
2. Let $S_1 = S$ and $i = 1$.
3. Discard sets in S_i where a literal and its complement appear to obtain S_i'.
4. Let T_i be the set of parent clauses in S_i' where p_i or $\neg p_i$ appears.
5. Let U_i be the set of resolvent clauses obtained by resolving (over p_i) every pair of clauses $C \cup \{p_i\}$ and $D \cup \{\neg p_i\}$ in T_i.
6. Set S_{i+1} to be $(S_i' \setminus T_i) \cup U_i$.
7. Set $i \leftarrow i + 1$, if $i = n + 1$ return S_i (S_{i+1} from previous step).
8. Otherwise return at step 3.

If S_{i+1} is the empty clause (resolution of $\{p\}$ with $\{\neg p\}$) then the set is unsatisfiable and thus the conclusion holds. If S_{i+1} is the empty set, i.e. containing no clauses (obtained by getting $S_i' = \emptyset$ by removing each clause), then the set is satisfiable and the conclusion does not necessarily hold. Note DPP always returns one of the empty clause or the empty set.

Theorem: If $S \vdash_r \{\}$ by DPP, then S is not satisfiable.

Proof. Use induction on i to show that each clause $C_i \in S_i$ has a resolution derivation from the initial set S (i.e. propagates forward). Since the output is the empty clause $\{\}$, there is a resolution from S to $\{\}$. Since $\{\}$ is not satisfiable and resolution preserves satisfiability by its soundness, S is not satisfiable. □

Theorem: If S is not satisfiable then $S \vdash_r \{\}$

Proof. Suppose for the sake of contradiction that the output is $\emptyset$ rather than $\{\}$ (only other case). If $S_{n+1} = \emptyset$ then S_{n+1} is vacuously satisfiable.

Notice S_{i+1} has variables $p_{i+1}, \ldots, p_n$ and S_i has variables $p_i, p_{i+1}, \ldots, p_n$. Recall that $S_{i+1} = (S_i' \setminus T_i) \cup U_i$. Suppose S_{i+1} is satisfied by the truth valuation t_{i+1} which thus satisfies U_i and $(S_i' \setminus T_i)$. Notice $S_i' = (S_i' \setminus T_i) \cup T_i$, hence to show S_i' it suffices to show T_i is satisfiable.

We show one of the following satisfies T_i, t_0 which agrees with t_{i+1} and has $p_i^{t_0} = 0$ or t_1 which agrees with t_{i+1} and has $p_i^{t_1} = 1$. Suppose for the sake of contradiction that neither of these is true. Notice t_0 satisfies all formulae which contains $\neg p_i$, hence it must falsify some $D \cup \{p_i\}$ in T_i. This means it is not satisfied by t_{i+1}. Similarly, we have there is some $E \cup \{\neg p_i\}$ which is not satisfied by t_1 and thus not satisfied by t_{i+1}. Since t_{i+1} does not satisfy D or E, it cannot satisfy $D \cup E$ which is a contradiction since $D \cup E \subseteq S_{i+1}$ and $S_{i+1}^{t_{i+1}} = 1$. Hence T_i is satisfiable.

Since T_i is satisfied, so is S_i'. Further, since S_i is S_i' with tautologies, this implies S_i is satisfiable. Thus S_{i+1} being satisfiable implies S_i is satisfiable.

We had that $S_{n+1} = \emptyset$ which is satisfiable, since satisfiability propagates backwards, S_1 is satisfiable a contradiction. Hence we must have $S \vdash_r \{\}$. $\square$

Remark: Since $\Sigma \cup \{\neg A\}$ is not satisfiable implies that $\Sigma \models A$, we have that formal resolution with DPP is sound. Since $\Sigma \models A$ implies $\Sigma \cup \{\neg A\}$ is not satisfiable, we have that DPP is complete. Notice also that DPP can prove invalidity as well.

5.2 First Order Logic

Definition. Domain: Also called the universe of discourse, it is the (non-empty to avoid triviality) collection of all objects (such as people, symbols, etc.) that affect a logical argument. Elements of a domain are called individuals or objects. Each individual is uniquely identified by an *individual symbol.* Properties of individuals are relations, also called predicates. A list of individuals with relations describing them is called an argument list.

Definition. Relations: A relation describes a property of an individual or between individuals. It is given a name and followed by a list of arguments. For instance expression Joan is the mother of Mary could be represented $mother(Joan, Mary) = M(j, m)$. Notice order is important. The number of arguments in the list is called the arity and cannot change. Notice a relation of arity n is called an n-ary relation. A unary relation is called a property. Relations typically use variables to describe the arguments without associating them with a particular individual.

Definition. Atomic Formula: A relation name followed by its argument is called an atomic formula. An atomic formula has a true or false value and can be combined by logical connectives like propositions. For instance if Socrates is human then Socrates is mortal becomes $Human(Socrates) \implies Mortal(Socrates)$. Since atomic formulae with arguments from the domain must have true/false values, they can be represented by a table. Notice further that binary relations often are written in infix notation, e.g. $a > b$, $c < d$, etc. rather than prefix notation, e.g. $\geq (e, f)$.

Remark: Formulae may also be given names, for instances we may write $A = Human(Socrates) \implies Mortal(Socrates)$.

Definition. Universal Quantifier: Let $A(u)$ represent a formula. If we want to indicate that $A(u)$ is true for all possible values of u in the domain, we write $\forall x A(x)$. $\forall x$ is the universal quantifier (pronounced for all) and $A(x)$ is called its scope. Notice here x is said to be bound by the quantifier.

Definition. Existential Quantifier: Let $A(u)$ represent a formula. If we want to indicate that $A(u)$ is true for at least one (possibly more) u in the domain, we write $\exists x A(x)$. $\exists x$ is the existential quantifier (pronounced there exists) and $A(x)$ is called its scope. Notice here x is said to be bound by the quantifier.

Remark. Bound and Free Variables: The variable appearing in a quantifier is said to be bound. For instance in $\forall x(P(x) \to Q(x))$ the bound variable x appears 3 times. A variable that is not bound is said to be free. A bound variable is bound to its scope, i.e. $P(x) \to Q(x)$, it is a placeholder for some other item. Free variables are used when wishing to refer to an arbitrary item, such as in defining relations.

Remark: Quantifiers are treated as unary connectives. They are also given higher precedence than all binary connectives. For instance, $\forall x(P(x) \lor Q(x))$ the $\forall$ takes priority.

Remark: Quantifiers can be nested, but they do not commute. For instance, $\forall x \in \mathbb{N}, \exists y \in \mathbb{N}, x < y$ is true and means that each natural number is smaller than another natural number. However $\exists y \in \mathbb{N}, \forall x \in \mathbb{N}, x < y$ is not true, it would imply there is a largest natural number. In first order logic symbols, with $\mathbb{N}$ being the domain it would be $\forall x \exists y(x < y)$ and $\exists y \forall x(x < y)$ respectively.

Remark. Negation of Quantifiers: Note that $\neg(\forall x P(x)) \models\mathrel{\mkern-5mu}\dashv \exists x(\neg P(x))$ and similarly $\neg(\exists x P(x)) \models\mathrel{\mkern-5mu}\dashv \forall x(\neg p(x))$.

Remark: In the case where the domain is finite, say $D = \{a_1, \ldots, a_n\}$, then we have $\forall x P(x) \models\mathrel{\mkern-5mu}\dashv P(a_1) \land \cdots \land P(a_n)$ and $\exists x P(x) \models\mathrel{\mkern-5mu}\dashv P(a_1) \lor \cdots \lor P(a_n)$.

Week 6 First-Order Logic Syntax and Semantics

6.1 First-Order Logic Syntax

Remark: Unlike in propositional logic, first-order logic does not use a language, rather it has two kinds of symbols. First are the logical symbols with fixed syntactic use and semantic meaning, such as the connectives of propositional logic. Second are the non-logical symbols or parameters, these have a designated syntax, but no pre-defined meaning.

Remark. Language of First-Order Logic: The formal language of first-order logic, denoted $\mathcal{L}$, consists of expressions using the following basic symbols:

Logical Symbols:

- connectives: $\neg, \lor, \land, \to, \leftrightarrow$
- free variable symbols: $u, v, w, u_1, \ldots$
- bound variable symbols: $x, y, z, x_1, \ldots$
- quantifiers: $\forall, \exists$
- punctuation symbols: ( ) ,

Non-logical symbols (generically):

- individual or constant symbols: $a, b, c, a_1, a_2, \ldots$, make up the domain
- relation or predicate symbols: $F, G, H, F_1, \ldots$
- function symbols: $f, g, h, f_1, \ldots$

Note: Note that each relation symbol and function symbol has an arity, the natural number representing its number of arguments. Note also there is a special binary relation called the equality symbol denoted $\approx$. $\mathcal{L}$ may or may not contain $\approx$, and when it does we call it the first-order language *with equality.*

Definition. Terms over $\mathcal{L}$: $Term(\mathcal{L})$ is the smallest class of expressions of $\mathcal{L}$ closed under the following formation rules: (1) Every individual symbol and (2) free-variable is a term of $\mathcal{L}$. (3) If $t_1, \ldots, t_n$ are terms of $\mathcal{L}$ and f is an n-ary function symbol, then $f(t_1, \ldots, t_n)$ is a term of $\mathcal{L}$. Note when $n = 2$ we may write this in infix notation, such as $t_1,\ f\ t_2$.

Definition. Closed Term: A term which contains no free variables is said to be a closed term.

Example. First-Order Language of Elementary Number Theory: This language has equality $=$, a binary relation symbol $<$ use for less-than, an individual symbol 0 used to denote zero, a unary function s called the successor, and two binary function symbols, $+$ for addition and $\times$ for multiplication.

This means $+(u, v)$ and $+(u, +(v, w))$ are terms, though they are commonly written as $(u+v)$ and $(u + (v + w))$. Note this means even though prefix notation is more correct, we often use infix notation for legibility.

Definition. FOL Atom: An expression of $\mathcal{L}$ is an atom in $Atom(\mathcal{L})$ if it is either of the form (1) $F(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n \in Term(\mathcal{L})$ and F is a relation symbol, or (2) $\approx (t_1, t_2)$ where $t_1, t_2 \in Term(\mathcal{L})$.

Definition. FOL Formula: $Form(\mathcal{L})$, the class of formulas of $\mathcal{L}$, is the smallest class of expressions of $\mathcal{L}$ closed under the following formation rules:

1. Every atom in $Atom(\mathcal{L})$ is a formula of $\mathcal{L}$.
2. If A is a formula of $\mathcal{L}$, then $(\neg A)$ is a formula of $Form(\mathcal{L})$.
3. If A, B are formulas of $\mathcal{L}$, then $(A \land B), (A \lor B), (A \to B), (A \leftrightarrow B)$ are formulas of $\mathcal{L}$.
4. If A is a formula of $\mathcal{L}$ with the free variable u and where x is a bound-variable which does not occur in $A(u)$, then $\forall x A(x)$ and $\exists x A(x)$ are formulas of $\mathcal{L}$. Note $A(x)$ denotes the expression formed from $A(u)$ by replacing every occurrence of u by x.

Remark:

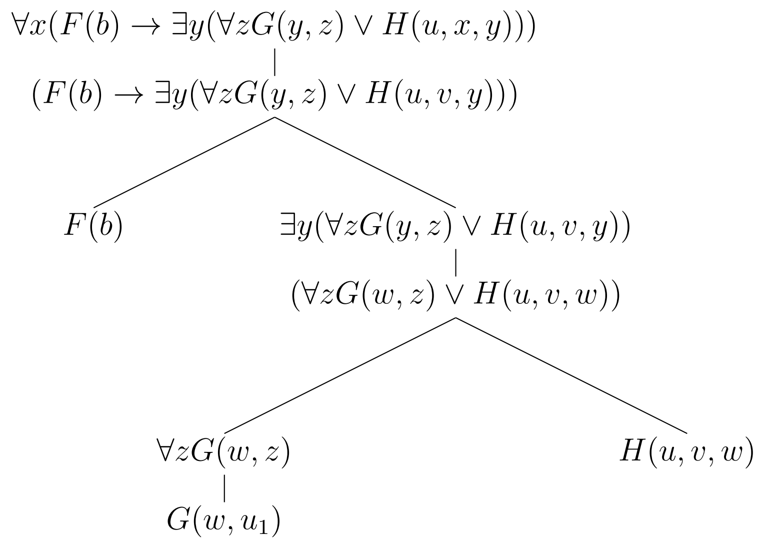
- Terms correspond to nouns and pronouns in English. They are expressions which name objects. They are built up of individual symbols and variables while applying any number of function symbols.
- Atoms are formed out of terms by applying exactly one relation symbol. These do not contain connectives or quantifiers. They are the comparison of the nouns.
- Formulas are expressions built from atoms using any number of connectives and quantifiers. They allow forming statements about the nouns, relating them and their properties to each other.

Theorem: Every term is of exactly one of the following forms: an individual symbol, a free variable symbol, or $f(t_1, \ldots, t_n)$ where f is an n-ary function symbol and $t_1, \ldots, t_n$ are terms.

Theorem: Every formula is of exactly one of the following forms: an atom, $(\neg A)$, $(A \land B)$, $(A \lor B)$, $(A \to B)$, $(A \leftrightarrow B)$, $\forall x A(x)$ or $\exists x A(x)$.

Remark: The proofs of the above theorems are analogous to the equivalent one in propositional logic.

Example. FOL Parse Tree: Consider the following Parse Tree for the formula $\forall x(F(b) \to \exists y(\forall z G(y,z) \lor H(u,x,y)))$


$\forall x(F(b) \to \exists y(\forall z G(y,z) \lor H(u,x,y)))$
$(F(b) \to \exists y(\forall z G(y,z) \lor H(u,v,y)))$
$F(b)$
$\exists y(\forall z G(y,z) \lor H(u,v,y))$
$(\forall z G(w,z) \lor H(u,v,w))$
$\forall z G(w,z)$
$H(u,v,w)$
$G(w,u_1)$

Definition. Sentence: A sentence or closed formula of $Form(\mathcal{L})$ is a formula of $\mathcal{L}$ in which no free variable symbol occurs. The set of sentences of $\mathcal{L}$ is denoted by $Sent(\mathcal{L})$.

Remark. Scoping: Quantifiers, like unary operators such as $\neg$, apply to the small scope possible.

Remark: When studying a specific type of mathematical structure or a specific field of mathematics, we often use a first-order language of 'X', where X is the studied structure. For instance, the first-order language of set theory.

6.2 First-Order Logic Semantics

Remark. Semantics of FOL Symbols: The connectives have the same meaning as in propositional logic. The quantifiers have had their meaning defined above (we also define their meaning more precisely below). The equality symbol denotes the relation of equality. The variables symbols will be understood as variables ranging over the domain.

Remark. Interpretation (Informally): An interpretation for $\mathcal{L}$ provides a non-empty set of objects called the domain and for each individual, relation, and function symbol of $\mathcal{L}$, a specification of the individuals, relations, and functions they denote. That is must specify an individual each individual symbol, provide a relation to each relation symbol, and a function over D for each function symbol.

Remark: In the case of $\mathcal{L}$ being associated with a specific structure (closed formulas) will thus express propositions about the structure.

Definition. Assignment: An assignment is a specification of a value in the domain for each free variable.

Definition. Valuation: A valuation is an interpretation and an assignment. We denote the meaning given by a valuation v to a symbol s by s^v or $v(s)$.

Definition. Relation: The graph of a n-ary relation, F, is a subset of $D^n = D \times D \times \cdot \times D$ given by $\{(x_1, \ldots, x_n) : F(x_1, \ldots, x_n) : x \in D\}$.

Definition. Graph: The graph of a n-ary function $f : D^n \to D$ is the relation $\{(x_1, \ldots, x_{n+1} : f(x_1, \ldots, x_n) = x_{n+1}, x_n, \ldots, x_{n+1} \in D\}$. A function is said to be total if each n tuple of elements in D maps to an element in D.

Example: The equality relation on D is $\{(x, x) : x \in D\}$. The graph of the summation function on the naturals is $\{(x, y, x + y) : x, y \in \mathbb{N}\}$.

Definition. Interpretation: An interpretation for the language of FOL $\mathcal{L}$ has a non-empty set called the domain, usually denoted D and a specification of an individual in D for each individual symbol, a relation over D to each relation symbol, and a function over D for each function symbol.

Thus a valuation v with an interpretation must have (1) for each individual constant symbol a, $a^v \in D$. (2) For each n-ary relation symbol F, the value F^v is an n-ary relation on D, i.e. $F^v \subseteq D^n$. (3) For each m-ary function symbol f, the value f^v is a total m-ary function from D^m to D.

Example: Proof there is a unicorn. Either there is an existing unicorn which exists or there is no existing unicorn which exists. Since the second is a contradiction the first must be true. Except the second is not in fact a contradiction. Let $E(x)$ denote x exists and D be the domain of unicorns. The first is $\exists x E(x)$, the second is $\neg\exists x E(x) \models\!\!\!\!\!\!=\!\!| \; \forall x \neg E(x)$. The second is true because because the domain is empty hence it is vacuously true. This is an example of why it is important to have a non-empty domain.

Example: Let A_1 be $F(c)$ where c is a constant (individual) and A_2 be $F(u)$ where u is a free variable. Consider and interpretation with domain $\mathbb{N}$ that interprets c as the number 3 and F as "is even". So A_1 has the value 0 whereas A_2 is undefined. If we then also specify an assignment where u is 2 for instance, then A_2 gets value 1.

Definition. Re-assignment: For any valuation v, free variable u, and domain element d, the valuation "v with u re-assigned to d" denoted $v(u/d)$ is given by

$$w^{v(u/d)} = \begin{cases} d & \text{if } w \text{ is } u \\ w^v & \text{if } w \text{ is not } u \end{cases}$$

Definition. Values: Let v be a valuation. For each term t, the value of t under v, denoted

t^v, is as follows.

- If t is a constant c, the value t^v is c^v.
- If t is a free variable u, the value t^v is u^v.
- If t is $f(t_1, \ldots, t_n)$, the value t^v is $f^v(t_1^v, \ldots, t_n^v)$

For each formula A, the value of A under v, denoted A^v, is as follows.

- If A is an atom $F(t_1, \ldots, t_n)$, then $A^v = 1$ if and only if $(t_1^v, \ldots, t_n^v) \in F^v$.
- If A is formed by a connective, the value A^v is determined the same way as in propositional logic.
- If A has the form $\forall x B(x)$, then

$$(\forall x B(x))^v = \begin{cases} 1 & \text{if } B(u)^{v(u/d)} = 1 \text{ for every } d \text{ in } D \\ 0 & otherwise \end{cases}$$

- If A has the form $\exists x B(x)$, then

$$(\exists x B(x))^v = \begin{cases} 1 & \text{if } A(u)^{v(u/d)} = 1 \text{ for some } d \text{ in } D \\ 0 & otherwise \end{cases}$$

(Note we assume u is a unique variable, hence it cannot appear in $\forall x A(x)$ or $\exists x A(x)$.)

Remark: The values $(\forall x A)^v$ and $(\exists x A)^v$ do not depend on any x^v, and in particular, since x is a bound variable, it has no value.

Definition. Satisfying: A valuation v satisfies a formula A if $A^v = 1$, a valuation v does not satisfy A if $A^v = 0$. A valuation v satisfies a set of formulae Σ if $A^v = 1$ for all $A \in \Sigma$, in this case we write $\Sigma^v = 1$.

Definition. Valid: A formula A is valid if every interpretation and valuation satisfy A, i.e. $A^v = 1$ for every v. A set of formulae $\Sigma \subseteq Form(\mathcal{L})$ is satisfiable if there is is an interpretation $\mathcal{I}$ such that $\Sigma^v = 1$. In this case we say $\mathcal{I}$ satisfies or is a model of Σ, or Σ is true in $\mathcal{I}$.

Definition. Satisfiable: A formula A is satisfiable if some interpretation and valuation satisfy A, i.e. $A^v = 1$ for some v. Otherwise it is said to be unsatisfiable.

Lemma: Let A be a first-order formula and v_1 and v_2 be valuations such that $u^{v_1} = u^{v_2}$ for every u that occurs free in A. Then $A^{v_1} = 1$ if and only if $A^{v_2} = 1$.

Proof. Use structural induction on the recursive creation of A. □

Definition. Logical Consequence: Let Σ be a set of formulae and A be a formula. Then A is the logical consequence of Σ, or Σ entails A, denoted $\Sigma \models A$ if for every valuation v, we have $\Sigma^v = 1$ implies $A^v = 1$. Notice this means A is valid if and only if $\emptyset \models A$.

Example: Show $\emptyset \models \forall x(A \to B) \to (\forall x A \to \forall x B)$. Prove by contradiction: suppose there is a valuation such that $((\forall x(A \to B)) \to ((\forall x A) \to (\forall x B)))^v = 0$. Then we have $(\forall x(A \to B))^v = 1$ and $((\forall x A) \to (\forall x B))^v = 0$, the latter implies $(\forall x A)^v = 1$ and $(\forall x A)^v = 0$. By definition of $\models$ for formulas with $\forall$ we have every a in the domain has $(A \to B)^{v(u/a)} = 1$ and $A^{v(u/a)} = 1$ thus $B^{v(u/a)} = 1$ for all a in the domain. Thus $(\forall x B)^v = 1$ and $(\forall x B)^v = 0$.

Example: Show that $\forall x \neg C(x) \models \neg \exists x C(x)$. Suppose $(\forall x(\neg C(x)))^v = 1$ for a valuation with domain D. By definition this means for every $a \in D$, $(\neg C(u))^{v(u/a)} = 1$. Thus for every $a \in D$, $C(u)^{v(u/a)} = 0$. Therefore, there is no $a \in D$ such that $C(u)^{v(u/a)} = 1$ which means $(\neg(\exists x C(x)))^v = 1$.

Example: Show that in general $(\forall x A \to \forall x B) \not\models \forall x(A \to B)$. Let $A(u)$ be $F(u)$. Let v have domain $\{a, b\}$ and $F^v = \{a\}$. Then $(\forall x A \to \forall x B)^v = 1$ for any B. Let B be $\neg F(u)$. Then $(\forall x(A \to B))^v = 0$ thus $(\forall x A \to \forall x B) \not\models \forall x(A \to B)$.

Example: Let $A(u)$ be any formula, let $A(t)$, for any term t, be the formula obtained by replacing ever instance of u in $A(u)$ by t. Show that $\emptyset \models \forall x A(x) \to A(t)$. If $(\forall x A(x))^v = 0$, there is nothing to prove, hence suppose $(\forall x A(x))^v = 1$. By definition we have $A(u)^{v(u/d)} = 1$ for every d in the domain, in particular, this holds for $d = t^v$. Thus we have $((\forall x A(x)) \to A(t))^v = 1$.

Remark: Validity is also called universal validity. This is since a valid formula is one that is true on account of its form alone. I.e. the truth of the formulae does not depend on the context: the interpretation of the formula, or specific values of variables. A satisfiable formula is one that is true for some particular context: for some specific interpretation and value assignment.

Remark: Valid formulas in $\mathcal{L}$ are the counter part of tautologies in $\mathcal{L}^p$, however they are stronger than tautologies. We can determine if a formula is a tautology using an algorithm, however this is not possible for valid formulas. This would require giving a value to $\forall x B(x)$ or $\exists x B(x)$ in finitely many steps which is not possible when there is an infinite domain.

Theorem: There is no algorithm for deciding the validity or satisfiability of formulas in $\mathcal{L}$. This was proved by Alonzo Church in 1936.

Remark. Higher-Order Logic: In first-order logic, variables and quantifiers range over individuals in the domain. In second-order logic, variables and quantifiers can also range over relation and function symbols. Hence we can talk about subsets of and relations on the domain. In higher-order logic, variables and quantifiers range over subsets, sets of sets, sets of sets of sets and so on arbitrarily.

Note. History: Aristotle (384-322 BCE) studied the earliest logic. George Boole (1815-1864) studied propositional logic. Gottlob Frege (1848-1925) studied predicate (first and second order) logic. Notice Euclid's Theorem that there are infinitely many primes cannot be expressed in propositional logic, but it can in predicate logic.

Second-order logic had been intended to express all of mathematics on firm logical foundation. However, Bertrand Russel (1872-1970) pointed out a paradox which made Frege's system inconsistent: Russell's Paradox.

Definition. Russell's Paradox: Let R be the set that contains all sets which are not members of themselves. Is R a member of itself?

Remark: Due to Russell's Paradox it has become uncommon to use second-order logic, using only first-order logic or more sophisticated systems has become the standard.

Week 7 Formal Deduction

Remark: Recall that where $A(u)$ is a formula containing u, we denote $A(x)$ to mean the quasi-formula which each variable u replaced by x. This becomes a formula if preceded by a quantifier. Where t is a term, we call $A(t)$ the substitution of u for t, which is where each variable u is replaced with t. The parse tree for $A(t)$ is the same as that of $A(u)$ but with the leaf containing u replaced by the whole parse tree t.

Remark: We wish to use natural deduction to extended the rules to first order logic using $\mathcal{L}$ instead of $\mathcal{L}^p$.

Definition. Rules of Formal Deduction:

1.	(Ref)	$A \vdash A$ is a theorem.
2.	$(+)$	If $\Sigma \vdash A$ is a theorem, then $\Sigma, \Sigma' \vdash A$ is a theorem.
3.	$(\neg-)$	If $\Sigma, \neg A \vdash B$ and $\Sigma, \neg A \vdash \neg B$ are theorems, then $\Sigma \vdash A$ is a theorem.
4.	$(\rightarrow -)$	If $\Sigma \vdash A \implies B$ and $\Sigma \vdash A$ are theorems, then $\Sigma \vdash B$ is a theorem.
5.	$(\rightarrow +)$	If $\Sigma, A \vdash B$ is a theorem, then $\Sigma \vdash A \implies B$ is a theorem.
6.	$(\wedge-)$	If $\Sigma \vdash A \wedge B$ is a theorem, then $\Sigma \vdash A$ and $\Sigma \vdash B$ are theorems.
7.	$(\wedge+)$	If $\Sigma \vdash A$ and $\Sigma \vdash B$ are theorems then $\Sigma \vdash A \wedge B$ is a theorem.
8.	$(\vee-)$	If $\Sigma, A \vdash C$ and $\Sigma, B \vdash C$ are theorems then $\Sigma, A \vee B \vdash C$ is a theorem.
9.	$(\vee+)$	If $\Sigma \vdash A$ is a theorem, then $\Sigma \vdash A \vee B$ and $\Sigma \vdash B \vee A$ are theorems.
10.	$(\leftrightarrow -)$	If $\Sigma \vdash A \iff B$ and $\Sigma \vdash A$ are theorems, then $\Sigma \vdash B$ is a theorem. If $\Sigma \vdash A \iff B$ and $\Sigma \vdash B$ are theorems, then $\Sigma \vdash A$ is a theorem.
11.	$(\leftrightarrow +)$	If $\Sigma, A \vdash B$ and $\Sigma, B \vdash A$ is a theorem, then $\Sigma \vdash A \iff B$ is a theorem.
12.	$(\forall-)$	If $\Sigma \vdash \forall A(x)$ is a theorem then $\Sigma \vdash A(t)$ is a theorem for all $t \in Term(\mathcal{L})$
13.	$(\forall+)$	If $\Sigma \vdash A(u)$ is a theorem and $u \notin \Sigma$, then $\Sigma \vdash \forall x A(x)$ is a theorem.
14.	$(\exists-)$	If $\Sigma, A(u) \vdash B$ is a theorem and $u \notin \Sigma$ and $u \notin B$, then $\Sigma, \exists x A(x) \vdash B$ is a theorem.
15.	$(\exists+)$	If $\Sigma \vdash A(t)$ is a theorem, then $\Sigma \vdash \exists x A'(x)$ is a theorem where $A'(x)$ results from $A(t)$ by replacing any number of occurrences of t by x
16.	$(\approx -)$	If $\Sigma \vdash A(t_1)$ and $\Sigma \vdash t_1 \approx t_2$ are theorems, then $\Sigma \vdash A'(t_2)$ is a theorem where $A'(t_2)$ results from $A(t_1)$ by replacing any number of occurrences of t_1 by t_2
17.	$(\approx +)$	$\emptyset \vdash u \approx u$ is a theorem

Example: Prove $\forall x P(x), \forall x(P(x) \to Q(x)) \vdash \forall x Q(x)$.

$$\begin{array}{lll}
\forall x P(x), \forall x(P(x) \to Q(x)) \vdash \forall x P(x) & (\in) & (1)\\
\forall x P(x), \forall x(P(x) \to Q(x)) \vdash \forall x(P(x) \to Q(x)) & (\in) & (2)\\
\forall x P(x), \forall x(P(x) \to Q(x)) \vdash P(u) & (\forall -)\ 1 & (3)\\
\forall x P(x), \forall x(P(x) \to Q(x)) \vdash P(u) \to Q(u) & (\forall -)\ 2 & (4)\\
\forall x P(x), \forall x(P(x) \to Q(x)) \vdash Q(u) & (\to -)\ 3,4 & (5)\\
\forall x P(x), \forall x(P(x) \to Q(x)) \vdash \forall x Q(x) & (\forall +)\ 5 & (6)
\end{array}$$

Example: Prove $\forall x(W(x) \to R(x)), \exists x W(x) \vdash \exists x R(x)$

$$\begin{array}{rll}
\forall x(W(x) \to R(x)), W(u) \vdash \forall x(W(x) \to R(x)) & (\in) & (1)\\
\forall x(W(x) \to R(x)), W(u) \vdash W(u) \to R(u) & (\forall -)\ 1 & (2)\\
\forall x(W(x) \to R(x)), W(u) \vdash W(u) & (\in) & (3)\\
\forall x(W(x) \to R(x)), W(u) \vdash R(u) & (\to -)\ 2,3 & (4)\\
\forall x(W(x) \to R(x)), W(u) \vdash \exists x R(x) & (\exists +)\ 4 & (5)\\
\forall x(W(x)) \to R(x)), \exists x W(x) \vdash \exists x R(x) & (\exists -)\ 5 & (6)
\end{array}$$

Definition. Formal Deducibility: For a set Σ of formulas in $\mathcal{L}$ and a formula $A \in \mathcal{L}$ we say A is formally deducible from Σ in predicate logic if the sequent $\Sigma \vdash A$ can be generated by the 17 rules of formal deduction.

Example: Prove $\neg \forall x A(x) \dashv\vdash \exists x \neg A(x)$

$$\begin{array}{rll}
\neg A(u) \vdash \neg A(u) & (\in) & (1)\\
\neg A(u) \vdash \exists x \neg A(x) & (\exists +)\ 1 & (2)\\
\neg \exists x \neg A(x) \vdash A(u) & (*)\ 2 & (3)\\
\neg \exists x \neg A(x) \vdash \forall x A(x) & (\forall +)\ 3 & (4)\\
\neg \forall x A(x) \vdash \exists x \neg A(x) & (*)\ 4 & (5)
\end{array}$$

(*) If $A \vdash B$ then $\neg B \vdash \neg A$ and possible use of $\neg\neg A \dashv\vdash A$.

$$\begin{array}{rll}
\forall x A(x) \vdash \forall x A(x) & (\in) & (1)\\
\forall x A(x) \vdash A(u) & (\forall -)\ 1 & (2)\\
\neg A(u) \vdash \neg \forall x A(x) & (*)\ 2 & (3)\\
\exists x \neg A(x) \vdash \neg \forall x A(x) & (\exists -)\ 3 & (4)
\end{array}$$

Lemma: Suppose $A \dashv\vdash A'$, $B \dashv\vdash B'$, $C(u) \dashv\vdash C'(u)$.

1. $\neg A \dashv\vdash \neg A'$
2. $A \wedge B \dashv\vdash A' \wedge B'$
3. $A \vee B \dashv\vdash A' \vee B'$

4. $A \to B \vdash\dashv A' \to B'$
5. $A \leftrightarrow B \vdash\dashv A' \leftrightarrow B'$
6. $\forall x C(x) \vdash\dashv \forall x C'(x)$
7. $\exists x C(x) \vdash\dashv \exists x C'(x)$

Theorem. Replacement of Equivalent Formulae: Let $A, B, C \in Form(\mathcal{L})$ with $B \vdash\dashv C$. Let A' result from A by substituting any number of occurrences of B by C. Then $A' \vdash\dashv A$.

Theorem. Complementation: Suppose A is a formula composed of atoms of $\mathcal{L}^p$, the connectives $\neg, \vee, \wedge$ and the two quantifiers by the formation rules concerned, and A' is the formula obtained by exchanging $\vee$ and $\wedge$, $\exists$ and $\forall$, and negating all atoms. Then $A' \vdash\dashv A$.

Theorem. Soundness and Completeness: Let $\Sigma \subseteq Form(\mathcal{L})$ and $A \in Form(\mathcal{L})$. Then $\Sigma \models A$ if and only if $\Sigma \vdash A$. I.e. formal deduction for predicate logic is sound and complete.

Week 8 Peano Arithmetic

8.1 Axiomatic Logic

Remark. Properties of Equality:

1. Reflexivity: $\forall x(x \approx x)$

$$\begin{array}{rll}
\emptyset \vdash u \approx u & \approx + & (1)\\
\emptyset \vdash \forall x(x \approx x) & \forall+\ (1) & (2)
\end{array}$$

2. Symmetry: $\forall x \forall y((x \approx y) \to (y \approx x))$

$$\begin{array}{rll}
\emptyset \vdash u \approx u & \approx + & (1)\\
u \approx v \vdash u \approx v & \in & (2)\\
u \approx v \vdash u \approx u & + & (3)\\
u \approx v \vdash v \approx u & \approx -\ (3),\ (2) & (4)\\
\emptyset \vdash (u \approx v) \to (v \approx u) & \to +\ (4) & (5)\\
\emptyset \vdash \forall y((u \approx y) \to (y \approx u)) & \forall+\ (5) & (6)\\
\emptyset \vdash \forall x \forall y((x \approx y) \to (y \approx x)) & \forall+\ (6) & (7)
\end{array}$$

3. Transitivity: $\forall x \forall y \forall z(((x \approx y) \wedge (y \approx z)) \to (x \approx z))$

$$\begin{array}{lll}
u \approx v \wedge v \approx w \vdash u \approx v \wedge v \approx w & \in & (1)\\
u \approx v \wedge v \approx w \vdash u \approx v & \wedge- & (2)\\
u \approx v \wedge v \approx w \vdash v \approx w & \wedge- & (3)\\
u \approx v \wedge v \approx w \vdash u \approx w & \approx -\ (2),\ (3) & (4)
\end{array}$$

$$\begin{array}{lll}
\emptyset \vdash (u \approx v \land v \approx w) \to u \approx w & \to+\ (4) & (5)\\
\emptyset \vdash \forall z((u \approx v \land v \approx z) \to u \approx z) & \forall+\ (5) & (6)\\
\emptyset \vdash \forall y \forall z((u \approx y \land y \approx z) \to u \approx z) & \forall+\ (6) & (7)\\
\emptyset \vdash \forall x \forall y \forall z((x \approx y \land y \approx z) \to x \approx z) & \forall+\ (7) & (8)
\end{array}$$

Theorem. EQSubs: Let u be a variable and let r, t_1, t_2 be terms. Let $r(v)$ denote r with all instances of u replaced by v. If $\Sigma \vdash t_1 \approx t_2$ then $\Sigma \vdash r(t_1) \approx r(t_2)$.

Proof.

$$\begin{array}{lll}
\emptyset \vdash r(t_1) \approx r(t_2) & \approx+ & (1)\\
\Sigma \vdash r(t_1) \approx r(t_2) & + & (2)\\
\Sigma \vdash t_1 \approx t_2 & \text{given} & (3)\\
\Sigma \vdash r(t_1) \approx r(t_2) & \approx-\ (2),(3) & (4)
\end{array}$$

□

Theorem. EQTrans: Let $t_1, \ldots, t_n$ be terms. If $\Sigma \vdash t_k \approx t_{k+1}$ for all $1 \leq k < n$ then $\Sigma \vdash t_1 \approx t_n$.

Proof. Proof by induction using transitivity of equality. □

Definition. Domain Axioms: A set A of domain axioms is a set of formulas which are accepted/assumed to always be true for a specific domain or class of domains.

- A should be decidable, i.e. there should be an algorithm to determine if a formula is a domain axiom.
- A should be sound with respect to the domain, i.e. if $A \vdash B$ then B should hold in the domain.
- A should be complete, i.e. it should be able to formally prove all true formulae of the domain.

Definition. Theory: A set of domain axioms together with all formulae they entail is called a theory. I.e. a theory is the set of all formulae that can be proven from a set of axioms.

Example. Euclid's Postulates: The earliest example of domain axioms appears in Euclid's Elements, these are Euclid's Postulates.

1. A straight line may be drawn between any two points.
2. Every straight line can be extended infinitely.
3. A circle may be drawn with any given point as the center, and any given radius.

4. All right angles are equal.
5. For every given point not on a given line, there is exactly one line through the point that does not meet the given line.

Note that the fifth postulate (parallel postulate) cannot be proved from the other four and there are interpretation where the first four hold but the fifth does not. For instance, in spherical geometry there is no such line and in hyperbolic geometry there are infinitely many lines.

Remark: We would like to find a small set of axioms from which all theorems about natural numbers follow (we consider 0 to be a natural number). These are the Peano Axioms.

Definition. Peano Axioms: The Peano Axioms consider three non-logical symbols, the constant 0, the functions $+$, $\cdot$, S (successor), and equality (we use $=$ and $\approx$ interchangeably). Recall the successor is such that $0 \mapsto 0$, $S(0) \mapsto 1$, $S(S(0)) \mapsto 2$, etc. The axioms are as follows:

PA1 $\forall x(S(x) \not\approx 0)$

PA2 $\forall x \forall y((S(x) \approx S(y)) \to (x \approx y))$

PA3 $\forall x(x + 0 \approx x)$

PA4 $\forall x \forall y(x + S(y) \approx S(x + y))$

PA5 $\forall x(x \cdot 0 \approx 0)$

PA6 $\forall x \forall y(x \cdot S(y) \approx x \cdot y + x)$

PA7 For each formula $A(u)$ with free variable u, $(A(0) \wedge \forall x(A(x) \to A(S(x)))) \to \forall x A(x)$ (Note this is in fact an infinite set is rather called an axiom schema.)

We denote $PA = \{PA1, \ldots, PA7\}$ the set of axioms. Note that PA is implicitly considered a premise of each theorem we prove in Peano arithmetic.

Notation: Given a theory T with a set of axioms X_T, we use $\Sigma \vdash_T C$ to denote $\Sigma, X_T \vdash C$.

Example: For Peano Arithmetic, we use $\Sigma \vdash_{PA} C$ to denote $\Sigma \cup PA \vdash C$.

8.2 Proofs in Peano Arithmetic

Example: Prove $\forall x(S(x) \not\approx x)$. Informally we want the base case $x = 0$. Notice by PA1 $\forall x(S(x) \not\approx 0)$, so for $x = 0$ the base case holds. Suppose BWOC that $S(S(x)) \approx S(x)$, then $S(x) \approx x$, a contradiction. Hence the inductive step holds. Formally:

$\emptyset \vdash_{PA} \forall x(S(x) \neq 0)$	PA1	(1)
$\emptyset \vdash_{PA} S(0) \neq 0$	$\forall-$ (1)	(2)
$S(u) \neq u, S(S(u)) = S(u) \vdash_{PA} S(S(u)) = S(u)$	$\in$	(3)
$\emptyset \vdash_{PA} \forall x \forall y(S(x) = S(y) \to x = y)$	PA2	(4)
$\emptyset \vdash_{PA} \forall x(S(x) = S(u) \to x = u)$	$\forall-$ (4)	(5)
$\emptyset \vdash_{PA} S(S(u)) = S(u) \to S(u) = u$	$\forall-$ (5)	(6)
$S(u) \neq u, S(S(u)) = S(u) \vdash_{PA} S(S(u)) = S(u) \to S(u) = u$	$+$ (6)	(7)
$S(u) \neq u, S(S(u)) = S(u) \vdash_{PA} S(u) = u$	$\to -$ (3) (7)	(8)
$S(u) \neq u, S(S(u)) = S(u) \vdash_{PA} S(u) \neq u$	$\in$	(9)
$S(u) \neq u \vdash_{PA} S(S(u)) \neq S(u)$	$\neg+$	(10)
$\emptyset \vdash_{PA} S(u) \neq u \to S(S(u)) \neq S(u)$	$\to +$ (10)	(11)
$\emptyset \vdash_{PA} \forall x(S(x) \neq x \to S(S(x)) \neq S(x))$	$\forall+$ (11)	(12)
$\emptyset \vdash_{PA} S(0) \neq 0 \land \forall x(S(x) \neq x \to S(S(x)) \neq S(x))$	$\land+$ (2) (12)	(13)
$\emptyset \vdash_{PA} (S(0) \neq 0 \land \forall x(S(x) \neq x \to S(S(x)) \neq S(x))) \to \forall x(S(x) \neq x)$	PA7 $S(x) \neq x$	(14)
$\emptyset \vdash_{PA} \forall x(S(x) \neq x)$	$\to -$ (13) (14)	(15)

Grouped by base case, inductive step, application of POMI (PA7).

Example: Prove $\forall x((x = 0) \lor \exists y(S(y) = x))$. Informally, we first prove the base case where $0 = 0$. We then prove using induction that if either $k = 0$ or $S(y) = k$ then either $S(k) = 0$ or there is a y where $S(y) = S(k)$. Let $P(u)$ denote the formula $(u = 0) \lor \exists y(S(y) = u)$. Formally:

$\emptyset \vdash_{PA} 0 = 0$	Reflexivity	(1)
$\emptyset \vdash_{PA} (0 = 0) \lor \exists y(S(y) = 0)$	$\lor+$ (1)	(2)
$\emptyset \vdash_{PA} S(k) = S(k)$	Reflexivity	(3)
$\emptyset \vdash_{PA} \exists y(S(y) = S(k))$	$\exists+$ (3)	(4)
$\emptyset \vdash_{PA} (S(k) = 0) \lor \exists y(S(y) = S(k))$	$\lor+$ (4)	(5)
$P(k) \vdash_{PA} P(S(k))$	$+$ (5)	(6)
$\emptyset \vdash_{PA} P(k) \to P(S(k))$	$\to +$ (6)	(7)
$\emptyset \vdash_{PA} \forall x(P(x) \to P(S(x)))$	$\forall+$ (7)	(8)
$\emptyset \vdash_{PA} P(0) \land \forall x(P(x) \to P(S(x)))$	$\land+$ (2) (8)	(9)
$\emptyset \vdash_{PA} (P(0) \land \forall x(P(x) \to P(S(x)))) \to \forall x P(x)$	PA7 $P(x)$	(10)
$\emptyset \vdash \forall x P(x)$	$\to -$ (9) (10)	(11)

Grouped by base case, inductive step, application of POMI (PA7).

Remark. More PA Properties:

- $\forall x(x \neq 0 \to \exists y(S(y) = x))$
- $\forall x \forall y(x + y = x \to y = 0)$

- $\forall x(x \cdot S(0) = x)$
- Associativity of addition: $\forall x \forall y \forall z((x + y) + z = x + (y + z))$
- Commutativity of addition: $\forall x \forall y(x + y = y + x)$ (requires double induction)
- Associativity of multiplication.
- Commutativity of multiplication.
- Distributivity: $\forall x \forall y \forall z(x \cdot (y + z) = x \cdot y + x \cdot z)$

Definition. More PA Relations: We further define additional relations to be used in Peano Arithmetic.

- $u \leq v$ if and only if $\exists z(u + z = v)$
- $u < v$ if and only if $\exists z(u + S(z) = v)$
- $Even(u)$ if and only if $\exists y(u = y + y)$
- $Prime(u)$ if and only if $(1 < u) \wedge \neg \exists y \exists z((u = y \cdot z) \wedge (1 < y) \wedge (1 < z))$

Example: Prove $\leq$ is transitive: $\forall x \forall y \forall z((x \leq y) \wedge (y \leq z) \to (x \leq z))$. This doesn't actually require induction. Informally, let u, v, w be arbitrary. Let $u + s = v$ and $v + t = w$. Then use EQTrans to replace w with $v + t$ and v with $u + s$. Then use $\forall-$ to eliminate free variables. Formally:

$$\begin{aligned}
u + s = v \wedge v + t = w &\vdash_{PA} u + s = v \wedge v + t = w && \text{Ref} && (1)\\
u + s = v \wedge v + t = w &\vdash_{PA} u + s = v && \wedge- \ (1) && (2)\\
u + s = v \wedge v + t = w &\vdash_{PA} v + t = w && \wedge- \ (1) && (3)\\
u + s = v \wedge v + t = w &\vdash_{PA} u + (s + t) = (u + s) + t && \text{Associativity} && (4)\\
u + s = v \wedge v + t = w &\vdash_{PA} (u + s) + t = v + t && \text{EQSub } (2) && (5)\\
u + s = v \wedge v + t = w &\vdash_{PA} u + (s + t) = w && \text{EQTrans } (4),(5),(3) && (6)\\
u + s = v \wedge v + t = w &\vdash_{PA} \exists k(u + k = w) && \exists+ \ (6) && (7)\\
u + s = v \wedge \exists k(v + k = w) &\vdash_{PA} \exists k(u + k = w) && \exists- \ (7) && (8)\\
\exists k(u + k = v) \wedge \exists k(v + k = w) &\vdash_{PA} \exists k(u + k = w) && \exists- \ (8) && (9)\\
\emptyset &\vdash_{PA} ((u \leq v) \wedge (v \leq w) \to (u \leq w)) && \to + \ (9) && (10)\\
\emptyset &\vdash_{PA} \forall z((u \leq v) \wedge (v \leq z) \to (u \leq x)) && \forall+ \ (10) && (11)\\
\emptyset &\vdash_{PA} \forall y \forall z((u \leq y) \wedge (y \leq z) \to (u \leq z)) && \forall+ \ (11) && (12)\\
\emptyset &\vdash_{PA} \forall x \forall y \forall z((x \leq y) \wedge (y \leq z) \to (x \leq z)) && \forall+ \ (12) && (13)
\end{aligned}$$

Remark: Using Peano Arithmetic, all theorems in number theory can be proved using the Peano Axioms and the rules of formal deduction. Peano Arithmetic is sound and thus also consistent by the soundness theorem. However, it is not complete, by Gödel's Incompleteness Theorem.

Theorem. Gödel's Incompleteness Theorem: In any consistent formal theory T with a decidable set of axioms, that is capable of expressing elementary arithmetic (e.g. Peano Arithmetic), there is a statement that is true but cannot be proven in T.

Week 9 FOL Resolution and Turing Machines

9.1 First Order Logic Resolution

Definition. Prenex Normal Form: (PNF) A formula is said to be in prenex normal form if it is of the form

$$Q_1x_1\ Q_2x_2 \cdots Q_nx_n\ B$$

where $n \geq 1$ and each Q_i is either $\forall$ or $\exists$ and the expression B is without quantifiers. The string $Q_1x_1 \cdots Q_nx_n$ is called the prefix and B the matrix. Where $n = 0$, the formula is considered a trivial case of prenex normal form.

Definition. Standardized Variables: The variables of a formula are said to be standardized if each distinct bound variable has a distinct symbol.

Theorem. Replaceability of Bound Variable Symbols: Let $A \in Form(\mathcal{L})$. Suppose A' results from A by replacing any number of occurrences of $QxB(x)$ with $QyB(y)$ where Q is either $\forall$ or $\exists$. Then $A \models\!\!\!\!\!\Dashv A'$ and $A \vdash\!\!\!\dashv A'$.

Remark. Algorithm to Convert to PNF: Each formula in $Form(\mathcal{L})$ is logically equivalent to a formula in PNF. The steps to find such a formula follow

1. Eliminate all instances of $\rightarrow$ and $\leftrightarrow$.
2. Move all negations inwards such that negations only appear as part of a literal (atom or its negation). De Morgan's laws and the formulae for negation of quantifiers.
3. Standardize the variables of the formula using the above theorem.
4. The formula in PNF can now be obtained by using the following equivalences:
   - $A \land \exists xB(x) \models\!\!\!\!\!\Dashv \exists x(A \land B(x))$, where x does not occur in A.
   - $A \land \forall xB(x) \models\!\!\!\!\!\Dashv \forall x(A \land B(x))$, where x does not occur in A.
   - $A \lor \exists xB(x) \models\!\!\!\!\!\Dashv \exists x(A \lor B(x))$, where x does not occur in A.
   - $A \lor \forall xB(x) \models\!\!\!\!\!\Dashv \forall x(A \lor B(x))$, where x does not occur in A.

Definition. $\exists$-free Prenex Normal Form: A sentence (formula without free variables) A is said to be in $\exists$-free prenex normal form if it is in prenex normal and does not contain an existential quantifier.

Remark: Consider a sentence of the form $\forall x_1 \cdots \forall x_n \exists yA$ with $n \geq 0$ such that A is an expression possibly with other quantifiers. Note that for each n-tuple $(a_1, \ldots, a_n)$ in the domain we find a y in the domain to satisfy A. Thus we can model y by a function $f(x_1, \ldots, x_n)$.

Definition. Skolem Function: For a sentence of the form $\forall x_1 \cdots \forall x_n \exists yA$ with $n \geq 0$, the function which consumes an n tuple in the domain and generates a satisfying y for A is called a Skolem function. That is, $\forall x_1 \cdots \forall x_n A'$ for all is satisfiable if and only if the above is where A' is A with $f(x_1, \ldots, x_n)$ replacing all instances of y. Note they are not tautologically equivalent since we assume the Skolem function to be well-defined. This therefore allows us to eliminate all existential quantifiers, this is called Skolemization.

Remark: We occasionally consider individual symbols as functions of zero arguments. This allows the Skolemization of sentences without universal quantifiers.

Remark. Algorithm to Convert to $\exists$ free PNF:

1. Transform the sentence A_0 to A_1 in PNF. Set $i = 1$.
2. Repeat until no existential quantifiers remain. Assume A_i is of the form $A_i = \forall x_1 \cdots \forall x_n \exists y A$ for some expression A. Set $A_i + 1 = A'$ where if $n = 0$, (no universal quantifiers) then A' is obtained from A by replacing all occurrences of y with c for some unique individual symbol c. If $n > 0$, A' is obtained from A by replacing all occurrences of y with $f(x_1, \ldots, x_n)$ for some unique function symbol f. Finally increment i. Note this process is from left to right.

Theorem: Given a sentence A in $Sent(\mathcal{L})$, there is an effective procedure for finding an $\exists$-free prenex normal form formula A' such that A is satisfiable if and only if A' is satisfiable.

Notation: After the existential quantifiers have been eliminated through Skolemization, it is customary to drop the universal quantifiers. We assume all variables to be universally quantified.

Theorem: Given a sentence A in $\exists$-free prenex normal form, one can effectively construct a finite set C_A of disjunctive clauses such that A is satisfiable if and only if the set C_A of clauses is satisfiable.

Theorem: Let Σ be a set of sentences and A be a sentence. The argument $\Sigma \models A$ is valid if and only if the set

$$C_{\neg A} \cup \left(\bigcup_{B \in \Sigma} C_B \right)$$

is not satisfiable.

Definition. Instantiation: An instantiation is an asissngment to a variable x_i of the quasi-term t'_i (one of individual symbol, variable symbol, or function symbol applied to individual or variable symbol). We write $x_i := t'_i$

Definition. Unification: Two formula in FOL are said to unify if there are instantiations that make the formulas in question identical. The act of unifying is called unification, the instantiation that unifies said formulas is called a unifier.

Note. Resolution for FOL: We aim to reach the empty clause $\{\}$ which is a contradiction. In propositional logic we need the same variable to occur more than once. In first order logic we need to get complementary literals. To obtain such literals we may need to use unification. (We still need complementary terms in each resolvent.)

Theorem: A set S of clauses in first order logic is not satisfiable if and only if there is a resolution derivation of the empty clause $\{\}$ from S. This means there is both soundness and completeness in resolution. If S is satisfiable then resolution with S may not output anything at all.

Example: Prove if everybody has a parent, everybody has a grandparent. Let the domain be the set of all people and $F(x, y)$ denote x is a parent of y. We wish to show

$$\forall x \exists y F(y, x) \models \forall x \exists y \exists z (F(z, y) \wedge F(y, x))$$

We generate the following set of clauses in $\exists$ free PNF:

$$\{F(f(x), x), \neg F(z, y) \vee \neg F(y, a)\}$$

Resolution is thus:

$$\begin{array}{llllr}
F(f(x), x) & \text{Premise} & & & (1) \\
\neg F(z, y) \vee \neg F(y, a) & \text{Negation of conclusion} & F(f(a), a) & (1) \text{ with } x := a & (2) \\
\neg F(z, f(a)) \vee \neg F(f(a), a) & (2) \text{ with } y := f(a) & & & (3) \\
\neg F(z, f(a)) & \text{Resolve } (3), (4) & & & (4) \\
F(f(f(a)), f(a)) & (1) \text{ with } x := f(a) & & & (5) \\
\neg F(f(f(a)), f(a)) & (2) \text{ with } z := f(f(a)) & & & (6) \\
\{\} & \text{Resolve } (6), (7) & & & (7)
\end{array}$$

Hence by the soundness of resolution our argument is valid.

Example: Consider the first two Peano axioms as a clause for resolution: (we replace $\approx$ with the prefix F relation)

$$\{\neg F(s(x), 0), \neg F(s(x'), s(y) \vee F(x', y)\}$$

trying to apply resolution to determine if this set is unsatisfiable, we get

$$\begin{array}{rrr}
\neg F(s(x), 0) & \text{Given} & (1) \\
\neg F(s(x'), s(y)) \vee F(x', y) & & (2) \\
\neg F(s(0), 0) & (1) \text{ with } x := 0 & (3) \\
\neg F(s(s(0)), s(0)) \vee \neg F(s(0), 0) & (2) \text{ with } x' := s(0),\ y := 0 & (4) \\
\neg F(s(s(0)), s(0)) & \text{Resolve } (3), (4) & (5) \\
\neg F(s(s(s(0))), s(s(0))) \vee \neg F(s(s(0)), s(0)) & (2) \text{ with } x' := s(s(0)), y := s(0) & (6) \\
\neg F(s(s(s(0))), s(s(0))) & \text{Resolve } (5), (6) & (7)
\end{array}$$

Clearly this resolution will never end, i.e. never reach the empty clause.

Remark: Misc. comments on resolution. In any clause we can remove duplicate terms. Resolutions of valid formulae should be avoided.

9.2 Turing Machines

Definition. Solvability: We say an algorithm solves a problem if for every input the algorithm produces the correct output. A problem itself is defined by specifying what the allowed inputs are and what constitutes a correct output for each input.

Remark: A Turing Machine is made up of a tape, consisting of cells one after another of any length. A cell being a container for a fixed amount of information. It also has a head located at a single cell that can read and write data at that cell and move to the next or previous cell. Further, it has a control unit that keep a current state and can read data via the head, then perform a single operation, and then write the result to the cell and change to a new state.

Definition. Turing Machine: A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ consists of

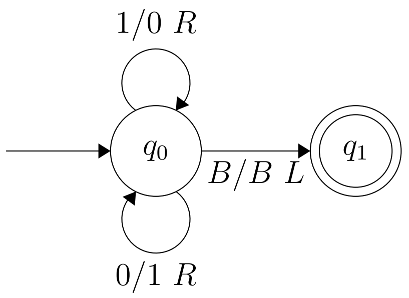
- Q is the *fixed* finite set of states of the control unit.
- Σ is the finite set ("alphabet") of input symbols.
- Γ is the finite set of tape symbols with $\Sigma \subseteq \Gamma$.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function, where L and R stand for the directions left and right respectively.
- $q_0 \in Q$ the start (initial) state.
- B is the blank symbol, $B \in \Gamma, B \notin \Sigma$.
- $F \subseteq Q$ is the set of final (accepting) states.

Note Q, Σ, Γ are unbounded, yet finite.

Initially, an input string in $\Sigma*$ (the set of all finite strings over Σ) is placed on the tape, one symbol per cell. All other cells contains blanks B. The computation begins in the control state q_0 and the head is placed at the first cell.

Remark: We often define transition functions by listing its values for each state and symbol. However, pictograms can also be used. For pictograms we use the convention that a state is a circle labelled with its name. The final state is doubly circled. A transition is an arrow from one state to another (possibly same) state. This arrow is labelled with a symbol to match to the one read, then a symbol to write to the cell, then a direction to proceed. The initial state is an arrow in from nowhere.

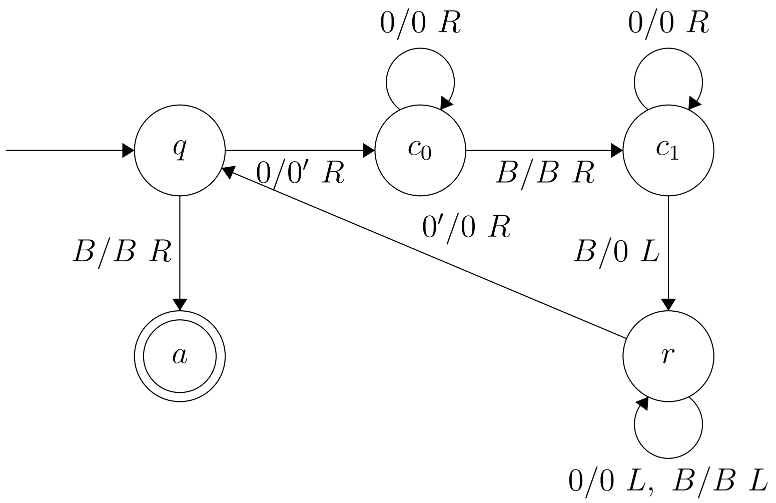
Example: The following transition inverts all the bits of a tape then ends on the last cell with data.



Definition. Halting: If a Turing machine M with input x reaches a state q and a symbol a such that $\delta(q, a)$ is undefined, then M halts on input x. If $q \in F$ then we say M accepts x, otherwise we say M rejects x. If M attempts to move left from the first (leftmost) cell on the tape, we say M crashes on input x. If neither of the above happen, M can continue making transitions forever. In this case, we say M runs forever on input x or loops on input x. We also say the final contents of the tape are M's output.

Example: Consider a machine whose input is n 0's in a row and we wish to copy them after the first blank. I.e. $0^n \to 0^n B 0^n$. We cannot use a different state for each 0 to keep track of how many we've, hence we need iterate more intelligently. Consider the following machine:

$Q = \{q, c_0, c_1, r, a\}$, $\Sigma = \{0\}$, $\Gamma = \{0, B, 0'\}$, start state q, accepting state a. The pictogram follows



Informally this machine changes a 0 if present to a $0'$ and then moves to the right until having seen two B's. Then it replaces the B with a 0 and moves to the left until reaching $0'$. At this point it changes the $0'$ to a 0 then moves on cell to the right. It stops upon reading a B.

Definition. Decidability: Let M be a Turing machine with input alphabet Σ. If for every input $x \in \Sigma$, M with input x halts, then M decides the language of strings over Σ that lead M to accept. That is it decides the set

$$\{x \in \Sigma* : M \text{ accepts input } x\}$$

A language is said to be decidable if there is a Turing machine that decides it. Otherwise, it is said to be undecidable.

Example: As shown previously, it is possible for a set of valid FOL to lead to endless resolution. Thus the set of valid FOL formulas is an undecidable language.

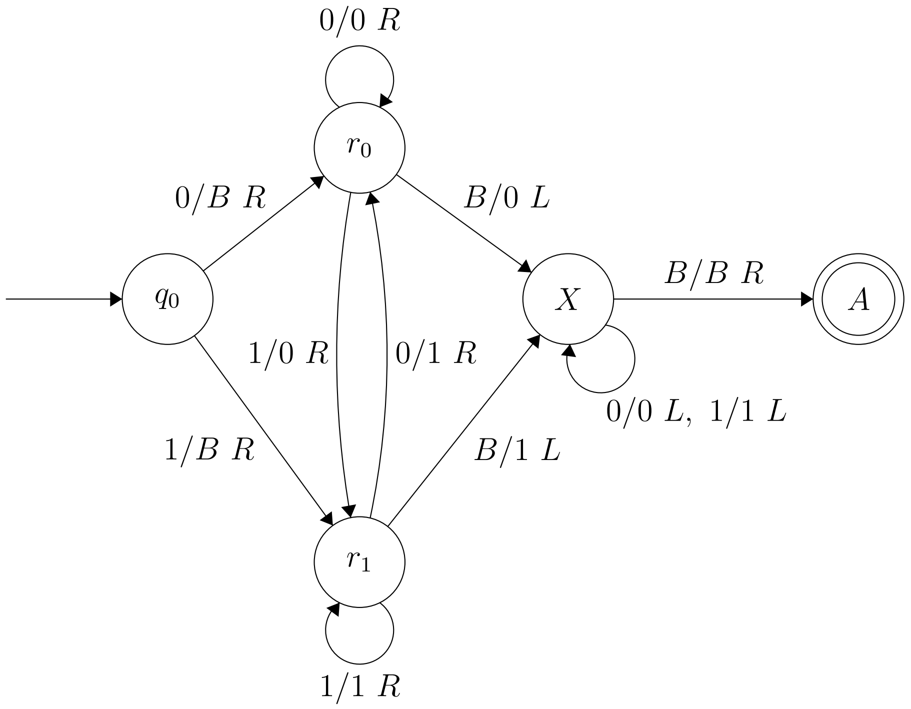
Unit 3 Turing Machines

Week 10 TM Undecidability and Reduction

10.1 Turing Machine Undecidability

Remark: Any computer program can be executed by a Turing machine. However, storage will look different. Modern computers use indexed storage whereas the Turing machine must use a tape. A Turing machine is nearly guaranteed to be slower, but just as capable.

Example. Turing Machine Shift: The following machine shifts the input data one cell to the right and returns the beginning



Remark: A Turing machine can store any fixed finite information in the control or in any cell. However, the size must not depend on the input. A Turing machine can make a non-constant amount of space on the tape using the above shift. It can also implement control structure to impose loops, subroutines (e.g. a shift), etc. We can form large Turing machines by combining smaller ones implemented as subroutines. These Turing machines begin to look more like flow charts.

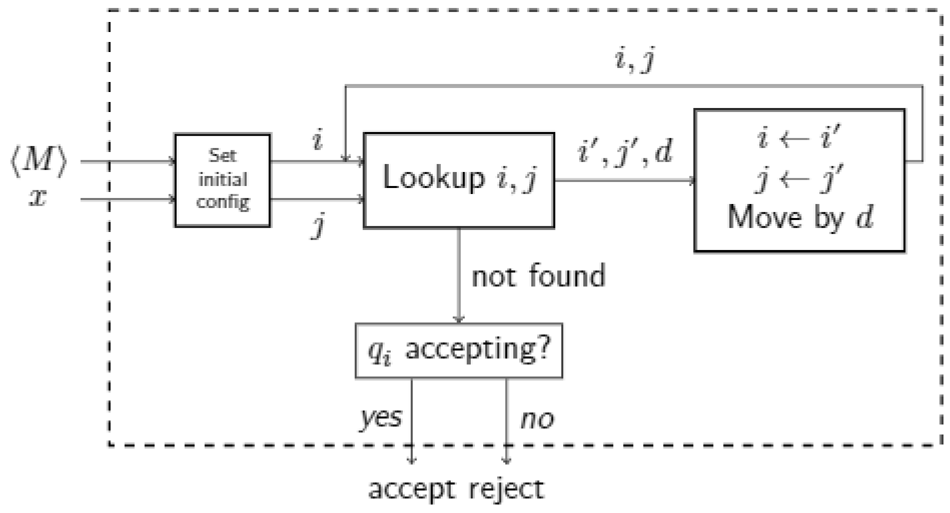
Definition. Universal Turing Machine: A universal Turing machine is a TM that given as input a description of a TM M and an input x for M, performs the actions of M on x and produces the same result. Essentially, the universal Turing machine accepts a program specified as a Turing machine and runs it on the input. This is analogous to an interpreter.

Note: How do we transform a TM into an input to another TM. Recall Q and Γ are finite sets. Their precise members aren't important to us as they server only to specify the transition function δ. Therefore, we simply assume $Q = \{q_0, \dots, q_k\}$ and $\Gamma = \{s_0, \dots, s_m\}$ for $k, m \in \mathbb{N}$. We always use q_0 to denote the initial state and s_0 to denote the blank.

The transition function δ can be specified by a set of 5-tuples of indices. We have a tuple (i, j, i', j', d) denote $\delta(q_i, s_j) = (q_{i'}, s_{j'}, d)$. Further, indices can be written in binary notation to limit the number of symbols needed.

If our alphabet has comma and parentheses (or other suitable symbols), we can simply concatenate all these tuples together to form a string containing all the information necessary to run M. Such a string is denoted $\langle M \rangle$ and called the code of M.

Note: Simulating M with input on x maintains three data areas. (1) $\langle M \rangle$, (2) the index of the current state of M, and (3) the current tape contents of M with a mark on the current symbol. To run the simulation, initialize the state to q_0 by making space, inserting 0, and marking the first symbol of x. We then look for the a 5-tuple (i, j, i', j', d) looking up on i, j. If such a tuple exists, update the state to i' and the marked symbol to j' and then move as specified by d. Otherwise, M halts. We check whether the state i is the index of a final state and accept or reject accordingly. Repeat until halting (if ever). See below for a pictogram.



Definition. Halting Problem: The Halting Problem is given $\langle M \rangle$ and x, does M with input x halt? As a language, we are asking the decidability of $HALT := \{(\langle M \rangle, x) : M \text{ halts on } x\}$. (Recall a machine decides a language if it accepts every member of the language and rejects all non-members.)

Theorem. Turing: The Halting Problem is undecidable.

Proof. By way of contradiction, suppose there is a machine D which decides halting. That is on input $(\langle M \rangle x)$ machine D halts and accepts if M halts on x and halts and rejects if M does not halt on x.

Now generate a new machine D' which modifies D. On input $\langle M \rangle$ run D with input $(\langle M \rangle, \langle M \rangle)$. Then if D accepted, D' runs forever, if D rejects then D' accepts. Now run D' with input $\langle D' \rangle$.

Therefore, if D' halts on input $\langle D' \rangle$, then D will accept $(\langle D' \rangle, \langle D' \rangle)$. Thus by construction D' runs forever on $\langle D' \rangle$, a contradiction.

If D' runs forever on input $\langle D' \rangle$, then D will accept $(\langle D' \rangle, \langle D' \rangle)$. Thus by construction D' will halt on $\langle D' \rangle$, a contradiction. □

10.2 Reducibility

Definition. Reduction: Suppose P_1 and P_2 are computational problems. A reduction from P_1 to P_2 is an algorithm (i.e. Turing machine) that always halts on any input and given and instance x_1 of P_1, produces and instance x_2 of problem P_2 such that in every case the answer to x_1 (as an instance of P_1) is the same as the answer to x_2 (as an instance of P_2). For a reduction R, we denote the output of to be $R(x)$.

Remark: A reduction allows us to solve another problem. For instance, suppose we can reduce P_1 to P_2 and we know how to solve P_2. Then to solve P_1 we reduce our input from P_1 to P_2 and then solve P_2 with input $R(x)$ (for $x \in P_1$) and return that answer.

Remark: A reduction allows us to prove another problem's undecidability. For instance suppose that we can reduce P_2 to P_1 and we know P_2 is undecidable. Then if P_1 were decidable, we could always reduce from P_2 to P_1 and then we would have that P_2 is decidable, a contradiction.

Theorem. Blank Tape Halting Problem: The problem, given a Turing machine M does M halt on empty input is undecidable.

Proof. Consider the reduction from a Turing machine M to the new machine M_w. If M_w has empty input, then we first write w to the tape and then move to the first position in the tape and continue execution of M with input w as usual. Thus M_w with empty input halts if and only if M halts with input w. Since the latter is undecidable by the halting problem, the former must be undecidable. The blank tape halting problem is therefore undecidable. □

Theorem. State-Entry Problem: The problem, given a Turing machine M, a state q and an input w, does M enter state q with input w is undecidable.

Proof. Consider the reduction from a Turing machine M to the new machine M'. Add a new state q to M and from all halting states of M, add a transition to q. So M halts on input w if and only if M' halts on state q on input w. Since the former is undecidable by the halting problem, the latter must be undecidable. The state-entry problem is therefore undecidable. □

Week 11 Undecidability and Program Verification

11.1 Undecideability

Remark: While we have thus far used Turing machines to motivate undecidability, this problem extends beyond the use of Turing machines.

Theorem. Post Correspondence Problem: The following problem is undecideable

> Given a finite sequence of pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ such that all s_i and t_i are strings of positive length, is there a sequence of indices $i_1, \ldots, i_n$ such that the concatenation $s_{i_1}, \ldots, s_{i_n}$ equals the concatenation $t_{i_1}, \ldots, t_{i_n}$.

Proof. One can make a reduction from the blank tape halting problem to PCP by making looking at the transitions of M. Therefore the PCP instance will have a solution if and only if M halts with empty input. □

Theorem. Hilbert's Tenth Problem: The following problem is undecideable (also called integer root problem)

> Given a polynomial $p(x_1, \ldots, x_n)$ with integer coefficients, does p have an integer root. I.e. are there integers $a_1, \ldots, a_n$ such that $p(a_1, \ldots, a_n) = 0$.

Remark: A proof, say by natural deduction, is correct if each deduction is correct. To verify that a deduction is correct we must simply determine if the sequents and a citation to the rule used are correct. This in turn can be verified by a Turing machine. Therefore, the problem given a list of formal sequents and citations of rules is the given list a formal proof is decidable.

Remark: The above claim can be made to use axioms as well, on the condition the set of axioms is decidable. That is, given a formula A and the claim that it is an axiom, we must be able to determine if the claim is correct.

Note: If a Turing machine M halts on input w, we can always prove that M will halt on w. This can be done by giving a transcript of the entire computation. Or this can be done in PA denoting a set of k symbols as numbers in base k. We can then use PA to treat these numbers as strings.

Lemma: There is a Turing machine that given a formula $A \in Form(\mathcal{L})$ outputs a proof of A if it exists. If no proof exists, the program may run forever with no output.

Theorem. Gödel's Incompleteness Theorem: Let Γ be a set of formulas, such that membership of Γ is decidable. Then either

1. $PA \cup \Gamma$ is inconsistent or
2. There is a formula B such that $PA \cup \Gamma \nvdash B$ and $PA \cup \Gamma \nvdash \neg B$

where PA is the set of Peano Axioms.

Proof. Suppose $PA \cup \Gamma$ is consistent. Consider the following Turing machine. On input $\langle M \rangle, x$. Let B denote the formula $\exists y(y$ is a proof that M halts on $x)$. Search for a proof of either $PA \cup \Gamma \vdash B$ or $PA \cup \Gamma \vdash \neg B$. If a proof of $PA \cup \Gamma \vdash B$ is found, halt with output "M halts on x." If a proof of $PA \cup \Gamma \vdash \neg B$ is found, halt with output "M does not halt on x."

This machine cannot halt on all formulas because if it did it would decide the halting problem. Thus there must be some M and x such that the program M does not halt on x. Thus B is an example of formula such that neither $PA \cup \Gamma \vdash B$ nor $PA \cup \Gamma \vdash \neg B$. □

Remark: Gödel's incompleteness theorem tells us that either a set of axioms is useless because Γ contradicts basic facts of arithmetic or Γ is undecidable – again it is useless, or there are facts about arithmetic are not implied by Γ. That is Γ does not completely capture all properties of the natural numbers.

Remark: Historically Gödel proved his theorem directly from the properties of arithmetic. As a key part of the proof he developed a coding of proofs as numbers (now referred to as Gödel numbering). Turing read the proof and realized he could adapt this to Turing machines, where he then proved the undecidability of the halting problem.

11.2 Program Verification

Remark: To prove that a program is correct it suffices to translate the specifications of the program to a formula in FOL, A_R, write a program which is meant to realize A_R, P, and then prove that P satisfies A_R.

Definition. Hoare Triple: A Hoare triple is of the form $(\!|P|\!)\ C\ (\!|Q|\!)$ where P is a pre-condition, a formula which is true before the program executes, Q is a post-condition, a formula which is true after the program executes, and C is a program.

Definition. Specification: A specification of a program C is a Hoare triple $(\!|P|\!)\ C\ (\!|Q|\!)$.

Example: Consider the specification "for a positive number x as input, compute a number whose square is less than x." This can be expressed as a Hoare triple $(\!|x > 0|\!)\ C\ (\!|y \cdot y < x|\!)$. Note that solutions need not be unique in behaviour or output. For instance the above has the solutions

```
y = 0;
```

and

```
y = 0;
while (y * y < x) {
    y = y + 1;
}
y = y - 1;
```

We could prevent the former solution from be valid by using a better postcondition such as $(y \cdot y < x) \wedge \forall z((z \cdot z < x) \to z \leq y)$.

Definition. Satisfied Under Partial Correctness: A Hoare triple $(\!|P|\!)\ C\ (\!|Q|\!)$ is satisfied under partial correctness, denoted

$$\models_{par} (\!|P|\!)\ C\ (\!|Q|\!)$$

if for every state s that satisfies P, the execution of C in state s terminates in state s' where s' satisfies Q.

Definition. Satisfied Under Total Correctness: A Hoare triple $(\!|P|\!)\ C\ (\!|Q|\!)$ is satisfied under partial correctness, denoted

$$\models_{tot} (\!|P|\!)\ C\ (\!|Q|\!)$$

if it is satisfied under partial correctness and the program terminates. We usually prove partial correctness and termination separately.

Example: The following Hoare triple is satisfied under total correctness:

```
(|x ≥ 0|)
y = 1;
z = 0;
while (z != x) {
    z = z + 1;
    y = y * z;
}
(|y = x!|)
```

However, if we set the pre-condition to be true the loop may repeat forever, thus making it only partially correct.

Example: The following Hoare triple is not satisfied under partial correctness

```
(|x ≥ 0|)
y = 1;
while (x != 0) {
    y = y * x;
    x = x - 1;
}
(|y = x!|)
```

This is because at the end of the loop, x has been changed, meaning $y \neq x!$ since $x = 0$.

Example: The following Hoare triple is satisfied under partial correctness for any pre-condition P and post-condition Q.

```
(|P|)
while (true) {
    x = 0;
}
(|Q|)
```

This is because the program never terminates.

Example: The following Hoare triple is satisfied under partial correctness for any program C

```
true
C
true
```

Notation: The following relations and function in FOL are commonly used:

- The relation $State(s)$ denotes s is a program state.
- The relation $Condit(P)$ denotes P is a condition.
- The relation $Code(C)$ denotes C is a program.
- The relation $Satisfies(s, P)$ denotes the state s satisfies the condition P.
- The relation $Terminates(C, s)$ denotes the program C terminates when execution begins in state s.
- The function $result(C, s)$ returns the state that results from executing code C beginning in state s if C terminates (undefined otherwise).

The domain is set of all program, program states, and conditions.

Remark. Partial and Total Correctness as FOL: Partial correctness of the Hoare triple $(\!|P|\!)\ C\ (\!|Q|\!)$:

$$\forall s(State(s) \to (Satisfies(s, P) \land Terminates(C, s) \to Satisfies(result(C, s), Q)))$$

Total correctness of the Hoare triple $(\!|P|\!)\ C\ (\!|Q|\!)$

$$\forall s(State(s) \to (Satisfies(s, P) \to Terminates(C, s) \land Satisfies(result(C, s), Q)))$$

Definition. Partial Correctness Proof: A partial correctness proof is annotated program with one or more conditions before and after each program statement. Each program statement (instruction) along with the following pre and post condition form a Hoare triple, where the pre-condition of one instruction is the post condition of the previous one and vice-versa. Further, each condition (aside from the first) has a justification as to why it holds.

Remark: We occasionally introduce logic variables (or auxillary variables) to make the proof easier to write.

Example: The following Hoare triple is satisfied under total correction.

```
(|x ≈ x_0 ∧ x_0 ≥ 0|)
y = 1;
while (x != 0) {
    y = y * x;
    x = x - 1;
}
(|y = x_0!|)
```

Definition. Inference Rule of Assignment:

$$\frac{}{(\!|Q[E/x]|\!)\ x = E;\ (\!|Q|\!)}\ (\text{assignment})$$

This rule is read as if the conditions above the Hoare triple are true, then the Hoare triple below holds. The stroke is read as in place of, that is if Q with all instances x replaced by E holds, then Q holds after `x = E;`.

Example: The following Hoare triple is satisfied under partial correctness by one application of the assignment rule.

$$(\!|y + 1 = 7|\!)\ \texttt{x = y + 1;}\ (\!|x = 7|\!)$$

Definition. Inference Rules about Implications: Implied Rule of "pre-condition strengthening":

$$\frac{P \to P' \quad (\!|P'|\!)\ C\ (\!|Q|\!)}{(\!|P|\!)\ C\ (\!|Q|\!)}\ (\text{implied})$$

Implied rule of "post-condition weakening":

$$\frac{(\!|P|\!)\ C\ (\!|Q'|\!) \quad Q' \to Q}{(\!|P|\!)\ C\ (\!|Q|\!)} l(\text{implied})$$

The first rule lets us assume more than we need, the second lets us conclude less than we can. That is whenever $P \to P'$ and $Q' \to Q$ then from $(\!|P'|\!)\ C\ (\!|Q'|\!)$ we can conclude $(\!|P|\!)\ C\ (\!|Q|\!)$. If an implication rule is used, we need to write a formal proof for $\emptyset \vdash P \to P'$ or $\emptyset \vdash Q' \to Q$. Usually we do these separately, after the proof of correctness.

Example: The following Hoare triple is satisfied under partial correctness

```
(|y = 6|)
(|y + 1 = 7|)   (implied)
x = y + 1;
(|x = 7|)   (assignment)
(|x ≤ 7|)   (implied)
```

Definition. Inference Rule for Composition of Instructions:

$$\frac{(\!|P|\!)\ C_1\ (\!|Q|\!) \qquad (\!|Q|\!)\ C_2\ (\!|R|\!)}{(\!|P|\!)\ C_1; C_2\ (\!|R|\!)}\ (\text{composition})$$

That is, if we can find a midcondition Q such that we can prove both $(\!|P|\!)\ C_1\ (\!|Q|\!)$ and $(\!|Q|\!)\ C_2\ (\!|R|\!)$, then we can bypass the midcondition and conclude $(\!|P|\!)\ C_1; C_2\ (\!|R|\!)$. The composition rule is often left implicit.

Remark: When constructing a formal proof of correctness, we often work bottom up. We start from the post-condition working our way to pre-condition line by line.

Week 12 Program Verification (cont.)

12.1 Program Verification (cont.)

Definition. Inference Rule for Conditionals: If-then rule

$$\frac{(\!|P \land B|\!)\ C\ (\!|Q|\!) \qquad (P \land \neg B) \to Q}{(\!|P|\!)\ \text{if (B) C}\ (\!|Q|\!)}\ (if - then)$$

If-then-else rule

$$\frac{(\!|P \land B|\!)\ C_1\ (\!|Q|\!) \qquad (\!|P \land \neg B|\!)\ C_2\ (\!|Q|\!)}{(\!|P|\!)\ \text{if (B)}\ C_1\ \text{else}\ C_2\ (\!|Q|\!)}$$

Example: A proof using if-then would follow along the lines of

```
(|P|)
if (B) {
    (|P ∧ B|) if-then
    C
    (|Q|)
}
(|Q|) if-then
```

where a sub-proof must show $(\!|P \land B|\!)\ C\ (\!|Q|\!)$. A proof with if-then-else follows along the lines of

```
(|P|)
if (B) {
    (|P ∧ B|)
    C_1
    (|Q|)
} else {
    (|P ∧ ¬B|)
    C_2
    (|Q|)
}
(|Q|)
```

where sub-proofs must show $(\!|P \land B|\!)\ C_1\ (\!|Q|\!)$ and $(\!|P \land \neg B|\!)\ C_2\ (\!|Q|\!)$ separately.

Example: Consider the following Hoare logic proof

```
(|true|)
if (max < x) {
    (|true ∧ max < x|) if-then
    (|x ≥ x|) implied (a)
    max = x;
    (|max ≥ x|)
}
(|max ≥ x|) if-then, implied (b)
```

Where for the last if-then we use (a) $\emptyset \vdash x \geq x$ by basic arithmetic the (b) fact that

$$(true \land \neg(max < x)) \to max \geq x$$

Example: Consider the following Hoare logic proof

```
⦇$true$⦈
if (x > y) {
    ⦇$x > y$⦈ if-then-else
    ⦇$(x > y \land x \approx x) \lor (x \leq y \land x \approx y)$⦈ implied (a)
    max = x;
    ⦇$(x > y \land max \approx x) \lor (x \leq y \land max \approx y)$⦈ assignment
} else {
    ⦇$\neg(x > y)$⦈ if-then-else
    ⦇$(x > y \land y \approx x) \lor (x \leq y \land y \approx y)$⦈ implied (b)
    max = y;
    ⦇$(x > y \land max \approx x) \lor (x \leq y \land max \approx y)$⦈ assignment
}
⦇$(x > y \land max \approx x) \lor (x \leq y \land max \approx y)$⦈ if-then-else
```

where (a) and (b) are obviously true by the reflexivity of equality.

Definition. Inference Rule Partial-while: The partial-while rule follows

$$\frac{(\!|I \land B|\!)\ C\ (\!|C|\!) I}{(\!|I|\!)\ \text{while (B) C}\ (\!|I \land \neg B|\!)}$$

Note that this rule does not require termination of the loop. Intuitively the loop will run until the while condition is not met (assuming no breaks are allowed) where I is kept as a loop invariant.

Definition. Loop Invariant: A loop invariant is a condition which is true both before and after each execution of the body of a loop and expresses a relationship among the variables used within the loop. It may or may not be useful in proving termination. It is often difficult to find a good loop invariant.

Example: A proof using partial-while would follow along the lines of

```
⦇$P$⦈
⦇$I$⦈ implied (a)
while (B) {
    ⦇$I \land B$⦈
    C
    ⦇$I$⦈
}
⦇$I \land \neg B$⦈
⦇$Q$⦈ implied (b)
```

Example: Consider the following incomplete Hoare logic proof

```
⦇$x \geq 0$⦈
y = 1;
z = 0;
while (z != x) {
    z = z + 1;
    y = y * z;
}
⦇$y \approx x!$⦈
```

The following is a sample trace of the variables

x	y	z
5	1	0
5	1	1
5	2	2
5	6	3
5	24	4
5	120	5

Notice that a candidate for the loop invariant could be $y \approx z!$. Loop invariants such as $y \geq z$ or $x \geq 0$ are not useful in concluding the post-condition or do not help us prove the loop-termination condition. We can complete this proof as

```
(| $x \geq 0$ |)
(| $1 = 0!$ |) implied (a)
y = 1;
(| $y = 0!$ |) assignment
z = 0;
(| $y \approx z!$ |) assignment
while (z != x) {
    (| $(y \approx z!) \wedge \neg(z \approx x)$ |) partial-while ((| $I \wedge B$ |))
    (| $y(z+1) \approx (z+1)!$ |) implied (b)
    z = z + 1;
    (| $yz \approx z!$ |) assignment
    $y = y * z$;
    (| $y = z!$ |) assignment
}
(| $y \approx z! \wedge z \approx x)$ |) partial-while ((| $I \wedge \neg B$ |))
(| $y = x!$ |) implied (c)
```

Implication (a) and (b) follow by definition of the factorial. Implication (c) follows by EQSubs and EQTrans.

Example: Consider the following incomplete Hoare logic proof

```
(| $n \geq 0 \wedge a \geq 0$ |)
s = 1;
i = 0;
while (i < n) {
    s = s * a;
    i = i + 1;
}
(| $s \approx a^n$ |)
```

A candidate for a loop invariant is $s \approx a^i$, however we run into the trouble that we get $s \approx a^i \wedge i \geq n$, how do we conclude $i = n$? For this it would be better instead to use the invariant $s \approx a^i \wedge i \leq n$. Our proof would then be

```
(| $n \geq 0 \wedge a \geq 0$ |)
(| $1 = a^0 \wedge 0 \leq n$ |) implied (a)
s = 1;
(| $s \approx a^0 \wedge 0 \leq n$ |) assignment
i = 0;
(| $s \approx a^i \wedge i \leq n$ |) assignment
while (i < n) {
    (|$s \approx a^i \wedge i \leq n \wedge i < n$|) partial-while
    (|$s \cdot a \approx a^{i+1} \wedge i+1 \leq n$|) implied (b)
    s = s * a;
    (|$s \approx a^{i+1} \wedge i+1 \leq n$|)
    i = i + 1;
    (|$s \approx a^i \wedge i \leq n$|)
}
(|$s \approx a^i \wedge i \leq n \wedge i \geq n$|) partial-while
(|$s \approx a^n$|) implied (c)
```

Remark. Proving Total Correctness: To prove total correctness we must show the program terminates, which requires finding a loop variant. A good choice is where the loop guard (condition required to keep looping) can be made to be an inequality with zero, e.g. turn $z < x$ into $0 < x - z$. When this loop *variant* keeps trending to zero, e.g. with $z \leftarrow z+1$ at the end of each loop we have $x - z \to 0$, we have the loop guard will eventually cause termination, as required.

Example: The followings proves the termination of the program

```
(|$x \geq 0$|)
y = 1;
z = 0;
                    At start of loop, $x - z = x \geq 0$
while (z != x) {
    z = z + 1;      $x - z$ decreases by 1.
    y = y * z;      $x - z$ is unchanged.
}
(|$y = x!$|)
```

Thus $x - z$ will eventually reach zero, wherein we have $x = z$ and the program terminates.

12.2 Hoare Logic & Arrays

Notation: We denote the elements of an array A of n elements by $A[0], A[1], \ldots, A[n-1]$.

Example: Notice our previous assignment rule falls short:

```
(|$A[y] \approx 0$|)
A[x] = 1;
(|$A[y] \approx 0$|)
```

This is false where $x = y$.

Definition. Array Assignment: We denote $A\{i \leftarrow e\}$ to mean the array with entries given by

$$A\{i \leftarrow e\}[j] = \begin{cases} e & \text{if } j = i \\ A[j] & \text{if } j \neq i \end{cases}$$

Definition. Inference rule of Array Assignment:

$$\frac{}{(\!|Q[A\{i \leftarrow e\}/A]|\!)\ \text{A[i] = e}\ (\!|Q|\!)}\ \text{array assignment}$$

Lemma: The following holds

$$A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[x] = A[y] \qquad \text{and} \qquad A\{x \leftarrow A[y]\}\{y \leftarrow A[x]\}[y] = A[x]$$

Proof. Consider the first equation. If $y \neq x$ the $\{y \leftarrow A[x]\}$ portion is redundant and so the claim holds. If $y = x$ then the left hand side is $A[x] = A[y]$. □