CS 4269/5469– Fundamentals of Logic In Computer Science

SEMESTER II, 2022-2023

Overall Notes

First-Order Logic

First-order Logic (FOL) is a formal language to describe and reason about *predicates* rather than propositions. A predicate is a proposition that depends on the value of some variables. To do this, first-order logic builds upon propositional logic with functions, variables, and quantification.

Motivation behind FOL

First-order logic grew out of the desire to study the foundations of mathematics in number theory and set theory. To illustrate the need for FOL, recall that we can represent the following statements as propositions in propositional logic:

$$s = \text{"Greeks are humans."}$$
$$r = \text{"Humans are mortals."}$$
$$p = \text{"Greeks are mortals."}$$

However, the limited expressiveness of propositional logic prevents us from reasoning about the elements in a universe. Thus, propositional logic cannot properly encode the following statements:

$s_1 =$ "If a person is a Greek, then he is a human." or $\forall x.\, G(x) \to H(x)$.
$s_2 =$ "There is a Greek." or $\exists x.\, G(x)$.
$s_3 =$ "There is a human." or $\exists x.\, H(x)$.

Here, G and H are *predicates* where $G(x)$ means x is a Greek and $H(x)$ means x is a human.

Syntax of FOL

We start by defining the syntax of first-order logic. First-order logic formulae are defined over a signature that identifies non-logical symbols, namely, the predicates, constants, and functions that can be used in formulae.

Definition 1 (Signature)**.** A *signature* or *vocabulary* is $\Sigma = (\mathcal{C}, \mathcal{F}, \mathcal{R})$ where

1. $\mathcal{C} = \{c_1, c_2, \ldots\}$ is a set of *constant symbols*.
2. $\mathcal{F} = \{f_1, f_2, \ldots\}$ is a set of *function symbols*. Each $f \in \mathcal{F}$ has an associated arity, denoted $\mathsf{arity}(f) \in \mathbb{N}_{\geq 0}$.
3. $\mathcal{R} = \{R_1, R_2, \ldots\}$ is a set of *relation symbols*. Each $R \in \mathcal{R}$ has an associated arity, denoted $\mathsf{arity}(R) \in \mathbb{N}_{>0}$.

Besides the signature, we also need a set $\mathcal{V} = \{x_1, x_2, \ldots\}$ of variables. We typically consider signatures Σ and variables $\mathcal{V}$ that are countable. Lastly, we also inherit the propositional connectives $\vee$ and $\neg$.

Now, we can define the set of *terms* in first-order logic.

Definition 2 (Terms)**.** A *terms* over a signature $\Sigma = (\mathcal{C}, \mathcal{F}, \mathcal{R})$ and variables $\mathcal{V}$ is given by the following BNF grammar:

$$t := x \mid c \mid f(t, t, \ldots, t)$$

where $x \in \mathcal{V}$, $c \in \mathcal{C}$, and $f \in \mathcal{F}$. The number of terms in a function f is determined by $\mathsf{arity}(f)$.

Having defined terms, we can use them to define well-formed formulae (wff) or formulae for short.

Definition 3 (Formulae)**.** A *well-formed formula (wff)* over a signature $\Sigma = (\mathcal{C}, \mathcal{F}, \mathcal{R})$ and variables $\mathcal{V}$ is given by the following BNF grammar:

$$\varphi := t = t \mid R(t, t, \ldots, t) \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\exists x.\, \varphi)$$

Here, t is a term, $x \in \mathcal{V}$ is a variable, and $R \in \mathcal{R}$ is a relation. The number of terms in a relation R is determined by $\mathsf{arity}(R)$.

We will additionally use the derived operators "$\wedge$", "$\rightarrow$" and the derived *universal* quantifier "$\forall$" which are obtained as follows:

1. $\varphi_1 \wedge \varphi_2 \equiv \neg((\neg\varphi_1) \vee (\neg\varphi_2))$
2. $\varphi_1 \rightarrow \varphi_2 \equiv (\neg\varphi_1) \vee \varphi_2$
3. $\forall x.\, \varphi \equiv \neg(\exists x.\, (\neg\varphi))$

From here on, when we say *formulae*, we really mean *well-formed formulae* in first-order logic. Furthermore, we will omit parantheses where the order of operations is clear. For example, we will write $\neg\varphi \vee \varphi_2$ instead of $(\neg\varphi) \vee \varphi_2$.

A formula φ is an *atomic formula* if it does not have any logical operators; that is, it is of the form $t_1 = t_2$ or $R(t_1, t_2, \ldots, t_k)$. Lastly, a *literal* is a formula that is either atomic or the negation of an atomic formula.

Example. Consider the signature $\Sigma = (\{c\}, \{f^1\}, \{<^2\})$ where f has arity 1 and $<$ has arity 2. Furthermore, suppose that $\mathcal{V} = \{x, y\}$. Then, the following are formulae:

1. $\exists x. \forall y. < (x, y) \vee x = y$
2. $\forall x. \forall y. x = y$
3. $\forall x. x = f(c)$
4. $x = f(c)$

Note that in formula 4, the variable x is not quantified. In this case, x is known as a *free variable*. On the other hand, all of the variables in formula 1 to 3 are quantified and thus, they are *bound variables*. Formulae with no free variables are known as *sentences*.

Semantics of FOL

The semantics of formulae in any logic is defined with respect to a *model*. In propositional logic, models were truth assignments to the propositions. For first-order logic, models are known as *structures* that help identify the interpretation of symbols in the signature.

Definition 4 (Structure)**.** Given a signature $\Sigma = (\mathcal{C}, \mathcal{F}, \mathcal{R})$, a *structure* $\mathcal{A}$ over Σ is a tuple (U, I) where:

- U is a non-empty set known as the *universe/domain* of the structure,
- For each constant symbol $c \in \mathcal{C}$, $I(c) \in U$ is its interpretation,
- For each function symbol $f \in \mathcal{F}$, $I(f)\colon U^{\mathsf{arity}(f)} \to U$ is its interpretation, and
- For each relation symbol $R \in \mathcal{R}$, $I(R) \subseteq U^{\mathsf{arity}(R)}$ is its interpretation.

The structure is *finite* if the universe U is finite.

Example. Consider the signature $\Sigma = (\{0\}, \{S^1\}, \{<^2\})$. Then, we have a Σ-structure $\mathcal{A} = (U, I)$ given by:

- $U = \mathbb{N}$,
- $I(0) = 0$,
- $I(s)(x) = x + 1$ for all $x \in U$, and
- $I(<) = \{(a, b) \mid a < b \text{ in the usual sense}\}$.

Now, consider the following formulae:

1. $\varphi_1 \equiv \exists x. < (0, x)$

2. $\varphi_2 \equiv <(0, x)$

As intuition leads, φ_1 is always true in $\mathcal{A}$. On the other hand, the truthiness of φ_2 depends on the *assignment* of x. This leads to following definition of assignments.

Definition 5 (Assignment)**.** For a Σ-structure $\mathcal{A}$, an *assignment* over $\mathcal{A}$ is a function $\alpha : \mathcal{V} \to U$ that assigns every variable $x \in \mathcal{V}$ a value $\alpha(x) \in U$.

We can then extend our choice of Σ-structure and assignment to a *valuation* function over the set of Σ-terms. A valuation gives an interpretation to the terms in the language.

Definition 6 (Valuation)**.** For a Σ-structure $\mathcal{A}$ and an assignment α over $\mathcal{A}$, a *valuation* $\mathsf{val}_{\mathcal{A},\alpha} : \mathsf{Terms} \to U$ is defined inductively as follows:

- For any variable $x \in \mathcal{V}$, $\mathsf{val}_{\mathcal{A},\alpha}(x) = \alpha(x)$
- For any constant $c \in \mathcal{C}$, $\mathsf{val}_{\mathcal{A},\alpha}(c) = I(c)$, and
- For any function $f^k \in \mathcal{F}$ where $k = \mathsf{arity}(f)$, we have

$$\mathsf{val}_{\mathcal{A},\alpha}(f(t_1, t_2, \ldots, t_k)) = I(f)(\mathsf{val}_{\mathcal{A},\alpha}(t_1), \mathsf{val}_{\mathcal{A},\alpha}(t_2), \ldots, \mathsf{val}_{\mathcal{A},\alpha}(t_k))$$

Before we finally define the entailment rules for formulae, we have to first define a new shorthand notation for *reassignment*. This helps us define the semantics of quantifiers.

Definition 7 (Reassignment)**.** For an assignment $\alpha : \mathcal{V} \to U$ over a Σ-structure $\mathcal{A} = (U, I)$, $\alpha[x \mapsto e]$ is the assignment

$$\alpha[x \mapsto u](y) = \begin{cases} \alpha(y) & \text{if } y \neq x \\ u & \text{otherwise.} \end{cases}$$

Finally, we can define the semantics of FOL formulae.

Definition 8 (Satisfaction)**.** Given a Σ-structure $\mathcal{A}$ and an assignment α, the *satisfaction* relation is a ternary relation $\models$. We write $\mathcal{A}, \alpha \models \varphi$ to mean that "φ holds in $\mathcal{A}$ under assignment α. We also write $\mathcal{A}, \alpha \not\models \varphi$ to mean that $\mathcal{A}, \alpha \models \varphi$ does not hold.

The satisfaction relation $\models$ is inductively defined as follows:

- $\mathcal{A}, \alpha \models t_1 = t_2$ iff $\mathsf{val}_{\mathcal{A},\alpha}(t_1) = \mathsf{val}_{\mathcal{A},\alpha}(t_2)$
- $\mathcal{A}, \alpha \models R(t_1, \ldots, t_k)$ iff $(\mathsf{val}_{\mathcal{A},\alpha}(t_1), \ldots, \mathsf{val}_{\mathcal{A},\alpha}(t_k)) \in I(R)$ where $k = \mathsf{arity}(R)$
- $\mathcal{A}, \alpha \models \neg\varphi$ iff $\mathcal{A}, \alpha \not\models \varphi$
- $\mathcal{A}, \alpha \models \varphi \vee \psi$ iff $\mathcal{A}, \alpha \models \varphi$ or $\mathcal{A}, \alpha \models \psi$
- $\mathcal{A}, \alpha \models \exists x.\, \varphi$ iff there exists $u \in U$ such that $\mathcal{A}, \alpha[x \mapsto u] \models \varphi$

We will now more formally define *bound* and *free* variables in a formula. We start by defining the *scope* of a quantifier.

Definition 9 (Scope)**.** Given a formula $\varphi = \exists x.\psi$, ψ is said to be the *scope* of the quantifier $\exists x$.

Definition 10 (Bound and Free Variables)**.** Every occurrence of the variable x in $\varphi = \exists x.\psi$ is called a *bound occurrence* of x in ψ. Any occurrence of x which is not bound is called a *free occurrence* of x in ψ.

The free variables in φ is denoted by $\mathsf{FVar}(\varphi)$. We go with the convention that a free variable of φ must occur in φ and thus $|\mathsf{FVar}(\varphi)|$ is bounded by the size of the formula φ.

The set of bound and free occurrences of a variable $x \in \mathcal{V}$ in a formula can be defined formally using induction on the structure of the formula, but is skipped here.

Notice that a variable may occur both bound and free. For example, in the formula $\varphi_1 \equiv R(\mathbf{x}) \to \forall x.\, T(\underline{x}, f(\underline{x}))$, the bolded variable is free but the underlined variables are bound. We can *rename* bound variables such that (a) bound variables are disjoint from free variables and (b) two bound occurrences of a variable refer to the same quantifier.

Claim 1. *For every formula φ, there is an equivalent formula ψ such that the bound and free variables of ψ are disjoint and every bound variable is in the scope of a unique quantifier.*

Example. The formula $\varphi_2 \equiv R(\mathbf{x}) \to \forall y.\, T(\underline{y}, f(\underline{y}))$ is equivalent to φ_1.

Definition 11 (Sentence)**.** A *sentence* is a formula φ that has no free variables (i.e. $\mathsf{FVar}(\varphi) = \varnothing$).

An analogous notion of the Relevance Lemma for FOL is the observation that the satisfaction of a formula depends only on the values that α assigns to the free variables of φ; the values assigned to bound variables are irrelevant. Before we prove the Relenvace Lemma for FOL, we first prove a related result on terms.

Lemma 1 (Relevance Lemma on Terms)**.** *Let t be a term and let $\mathcal{A} = (U, I)$ be a structure. If assignments α_1 and α_2 are such that $\alpha_1(x) = \alpha_2(x)$ for each variable x occurring in t, then $\mathsf{val}_{\mathcal{A},\alpha_1}(t) = \mathsf{val}_{\mathcal{A},\alpha_2}(t)$.*

Proof. We will prove by induction on the structure of t that if $\alpha_1(x) = \alpha_2(x)$ for each variable x occurring in t, then $\mathsf{val}_{\mathcal{A},\alpha_1}(t) = \mathsf{val}_{\mathcal{A},\alpha_2}(t)$.

Base Case 1. For any constant symbol $c \in \mathcal{C}$, $\mathsf{val}_{\mathcal{A},\alpha_1}(c) = I(c) = \mathsf{val}_{\mathcal{A},\alpha_2}(c)$ by definition.

Base Case 2. For any variable $x \in \mathcal{V}$, $\mathsf{val}_{\mathcal{A},\alpha_1}(x) = \alpha_1(x) = \alpha_2(x) = \mathsf{val}_{\mathcal{A},\alpha_2}(x)$ by the assumption.

Inductive Step. Assume that the inductive hypothesis holds for some terms $t_1, \dots t_k$ for some $k \in \mathbb{N}$. Consider any function symbol $f^k \in \mathcal{F}$ where $k = \text{arity}(f)$. Then, we have:

$$\begin{aligned}
\text{val}_{\mathcal{A},\alpha_1}(f(t_1,\dots,t_k)) &= I(f)(\text{val}_{\mathcal{A},\alpha_1}(t_1),\dots,\text{val}_{\mathcal{A},\alpha_1}(t_k)) && \text{(by definition)}\\
&= I(f)(\text{val}_{\mathcal{A},\alpha_2}(t_1),\dots,\text{val}_{\mathcal{A},\alpha_2}(t_k)) && \text{(by inductive hypothesis)}\\
&= \text{val}_{\mathcal{A},\alpha_2}(f(t_1,\dots,t_k)) && \text{(by definition)}
\end{aligned}$$

Thus, the inductive hypothesis holds for all terms t. □

Now, we can use the previous result to prove the Relevance Lemma on FOL formulae.

Lemma 2 (Relevance Lemma on Formulae)**.** *Let φ be a FOL formula and let $\mathcal{A}$ be a structure. If assignments α_1 and α_2 are such that $\alpha_1(x) = \alpha_2(x)$ for every $x \in \text{FVar}(\varphi)$, then $\mathcal{A}, \alpha_1 \models \varphi$ iff $\mathcal{A}, \alpha_2 \models \varphi$.*

Proof. Without loss of generality, suppose that $\mathcal{A}, \alpha_1 \models \varphi$. We want to show that $\mathcal{A}, \alpha_2 \models \varphi$. Once we have shown this, then by swapping α_1 and α_2 in our argument, we will have also proven that if $\mathcal{A}, \alpha_2 \models \varphi$ then $\mathcal{A}, \alpha_1 \models \varphi$.

So, to proceed, we will prove by induction on the structure of φ that if $\alpha_1(x) = \alpha_2(x)$ for every $x \in \text{FVar}(\varphi)$, and $\mathcal{A}, \alpha_1 \models \varphi$, then $\mathcal{A}, \alpha_2 \models \varphi$.

Base Case 1. For any atomic formula of the form $\varphi \equiv t_1 = t_2$ for some terms t_1 and t_2, we have:

$$\begin{aligned}
\mathcal{A}, \alpha_1 &\models \varphi && \text{(by assumption)}\\
\text{val}_{\mathcal{A},\alpha_1}(t_1) &= \text{val}_{\mathcal{A},\alpha_1}(t_2) && \text{(since } \mathcal{A}, \alpha_1 \models \varphi\text{)}\\
\text{val}_{\mathcal{A},\alpha_2}(t_1) &= \text{val}_{\mathcal{A},\alpha_2}(t_2) && \text{(by the previous lemma)}\\
\mathcal{A}, \alpha_2 &\models \varphi && \text{(by definition)}
\end{aligned}$$

Base Case 2. For any atomic formula of the form $\varphi \equiv R(t_1, \dots, t_k)$ for some terms $t_1, \dots, t_k$ where $k = \text{arity}(R)$, we have:

$$\begin{aligned}
\mathcal{A}, \alpha_1 &\models \varphi && \text{(by assumption)}\\
(\text{val}_{\mathcal{A},\alpha_1}(t_1),\dots,\text{val}_{\mathcal{A},\alpha_1}(t_k)) &\in I(R) && \text{(by definition)}\\
(\text{val}_{\mathcal{A},\alpha_2}(t_1),\dots,\text{val}_{\mathcal{A},\alpha_2}(t_k)) &\in I(R) && \text{(by the previous lemma)}\\
\mathcal{A}, \alpha_2 &\models \varphi && \text{(by definition)}
\end{aligned}$$

Inductive Step 1. Assume that the inductive hypothesis holds for some formula ψ. Consider the formula $\varphi \equiv \neg\psi$. Then, we have:

$$\begin{aligned}
\mathcal{A}, \alpha_1 &\models \varphi && \text{(by assumption)}\\
\mathcal{A}, \alpha_1 &\not\models \psi && \text{(by definition)}\\
\mathcal{A}, \alpha_2 &\not\models \psi && \text{(by the inductive hypothesis)}\\
\mathcal{A}, \alpha_2 &\models \varphi && \text{(by definition)}
\end{aligned}$$

Inductive Step 2. Assume that the inductive hypothesis holds for some formulae ψ_1 and ψ_2. Consider the formula $\varphi \equiv \psi_1 \vee \psi_2$. Then, we have:

$\mathcal{A}, \alpha_1 \models \varphi$	(by assumption)
$\mathcal{A}, \alpha_1 \models \psi_1 \vee \psi_2$	(by definition)
$\mathcal{A}, \alpha_1 \models \psi_1$ or $\mathcal{A}, \alpha_1 \models \psi_2$	(by definition)
$\mathcal{A}, \alpha_2 \models \psi_1$ or $\mathcal{A}, \alpha_2 \models \psi_2$	(by the inductive hypothesis)
$\mathcal{A}, \alpha_2 \models \varphi$	(by definition)

Inductive Step 3. Assume that the inductive hypothesis holds for some formula ψ. Consider the formula $\varphi \equiv \exists x. \psi$. Then, we have:

$\mathcal{A}, \alpha_1 \models \varphi$	(by assumption)
$\mathcal{A}, \alpha_1 \models \exists x. \psi$	(by definition)
$\exists u \in U.\ \mathcal{A}, \alpha_1[x \to u] \models \psi$	(by definition)
$\exists u \in U.\ \mathcal{A}, \alpha_2[x \to u] \models \psi$	(by the inductive hypothesis)
$\mathcal{A}, \alpha_2 \models \exists x. \varphi$	(by definition)
$\mathcal{A}, \alpha_2 \models \varphi$	(by definition)

Hence, we have proven that if $\mathcal{A}, \alpha_1 \models \varphi$, then $\mathcal{A}, \alpha_2 \models \varphi$. Since our argument does not assume any additional property about α_1 or α_2, by swapping α_1 and α_2 in our argument, we can conclude that if $\mathcal{A}, \alpha_2 \models \varphi$, then $\mathcal{A}, \alpha_1 \models \varphi$.

Therefore, we have proven that, assuming that $\alpha_1(x) = \alpha_2(x)$ for every $x \in \mathsf{FVar}(\varphi)$, then $\mathcal{A}, \alpha_1 \models \varphi$ iff $\mathcal{A}, \alpha_2 \models \varphi$. □

It follows from the Relevance Lemma that if φ is a sentence, then all variable assignments are equivalent with respect to satisfiability. Hence, for any sentence φ, we simply write $\mathcal{A} \models \varphi$ whenever $\mathcal{A}, \alpha \models \varphi$ for some assignment α.

Corollary 1. *For any sentence φ and assignments α_1 and α_2, $\mathcal{A}, \alpha_1 \models \varphi$ iff $\mathcal{A}, \alpha_2 \models \varphi$.*

Satisfiability and Validity of FOL Formulae

We can define the satisfiability and validity of FOL formulae similar to the way we defined them for propositional logic.

Definition 12 (Satisfiability)**.** A FOL formula φ over signature Σ is *satisfiable* if there is some structure $\mathcal{A}$ and assignment α such that $\mathcal{A}, \alpha \models \varphi$. Otherwise, φ is *unsatisfiable*.

Definition 13 (Satisfiability of a Set)**.** A set of FOL formulae Γ is *satisfiable* if there is a structure $\mathcal{A}$ and assignment α such that $\mathcal{A}, \alpha \models \varphi$ for every $\varphi \in \Gamma$. Equivalently, we write $\mathcal{A}, \alpha \models \Gamma$.

Definition 14 (Validity)**.** A FOL formula φ is said to be *valid* if for every structure $\mathcal{A}$ and assignment α, $\mathcal{A}, \alpha \models \varphi$.

Definition 15 (Logical Consequence)**.** A formula φ is a *logical consequence* of a set of formulae Γ if for each structure $\mathcal{A}$ and assignment α, $\mathcal{A}, \alpha \models \Gamma$ implies that $\mathcal{A}, \alpha \models \varphi$.

As a shorthand, when $\varnothing \models \varphi$, we write $\models \varphi$.

Definition 16 (Logical Equivalence)**.** Two formulae φ_1 and φ_2 are *logically equivalent* if for every structure $\mathcal{A}$ and assignment α, $\mathcal{A}, \alpha \models \varphi_1$ iff $\mathcal{A}, \alpha \models \varphi_2$.

Definition 17 (Equisatisfiability)**.** Two formulae φ_1 and φ_2 are *equisatisfiable* when φ_1 and φ_2 are both satisfiable or both unsatisfiable.

Primer on Countability

Definition 18 (Injection)**.** A function $f : A \to B$ is said to be *injective* if for all $a_1, a_2 \in A$, if $f(a_1) = f(a_2)$, then $a_1 = a_2$.

We aim to represent cardinality of sets in terms of injections.

Definition 19 (Countable set)**.** A set S is said to be *countable* if there exists an injection $f : S \to \mathbb{N}$.

Notice that if a set is countable, then one can assign indices which are natural numbers to the elements of the set such that no two elements of S get the same index.

This is analogous to the idea of "enumerating" the elements of S.

Claim 2 (Finite sets are countable)**.** *A finite set is countable.*

Proof sketch. Let e be an arbitary enumeration of S. Consider the function $f : S \to \mathbb{N}$ defined by $f(a) = i$ such that $i \in \mathbb{N}$ is the index of a in e; that is, $f(a) = e_i$. We can show that this is an injection □

To formally prove Claim 2, we will have to come up with an injective function from any finite set to the natural numbers. We will omit the proof here.

A more crucial observation is the following theorem.

Theorem 1. There are sets that are not countable; that is, they are *uncountable*.

Example. The set of real numbers $\mathbb{R}$ and the powerset of the natural numbers $\mathscr{P}(\mathbb{N})$ are uncountable.

Next, we will state a straightforward claim.

Claim 3. *The set of natural numbers $\mathbb{N}$ is countable.*

Proof sketch. Consider the function $f : \mathbb{N} \to \mathbb{N}$ defined by $f(n) = n$. We can show that this is an injection. □

Another simple observation is the following.

Claim 4. *Let S be a set and $S' \subseteq S$. If S is countable, then S' is countable.*

Proof sketch. Consider an injection $f : S \to \mathbb{N}$. We can extend f to an injection $g : S' \to \mathbb{N}$ by defining $g(s) = f(s)$ for all $s \in S'$. □

Example. Some other examples of countable sets are: the set of even numbers, odd numbers, and prime numbers. These are all subsets of the natural numbers.

Notice that while there is an injection from the set of even numbers to the set of naturals, there is also an injection from the set of naturals to the set of even numbers. We will formalise this notion in the following claim.

Claim 5. *If A and B are countably infinite sets, then there is a bijection between A and B. Thus, $|A| = |B|$.*

An even more important result is the following claim:

Claim 6. *Let A and B be countably infinite sets. Then, $A \times B = \{(a, b) \mid a \in A, b \in B\}$ is countable.*

The idea of a proof for Claim 6 is to draw a grid and list the elements of A along the rows and the elements of B along the columns. Numbering the elements along the diagonals of the grid, we can then define an injection from $A \times B$ to $\mathbb{N}$.

Claim 6 yields the following corollaries:

Corollary 2. *$\mathbb{N} \times \mathbb{N}$ is countable.*

Corollary 3. *$\mathbb{Q}$ is countable.*

Proof sketch. Every rational $r \in \mathbb{Q}$ is of the form p/q where $p, q \in \mathbb{N}$. Thus, r can be represented as a pair $(p, q) \in \mathbb{N} \times \mathbb{N}$ and so $\mathbb{Q}$ is isomorphic to a subset of $\mathbb{N} \times \mathbb{N}$. By Corollary 2 and Claim 4, $\mathbb{Q}$ is countable. □

Corollary 4. *$\mathbb{Z}$ is countable.*

Proof sketch. For every integer i, we can write it as either $(0, i)$ or $(-1, i)$. Thus, $\mathbb{Z}$ is isomorphic to a subset of $\mathbb{N} \times \mathbb{N}$. By Corollary 2 and Claim 4, $\mathbb{Q}$ is countable. □

Theorem 2. There is a collection $\mathcal{C}$ of countable sets such that $\mathcal{C}$ is uncountable.

Example. The powerset of the natural numbers $\mathscr{P}(\mathbb{N})$ is an uncountable collection of countable sets.

Next, we state a more important observation:

Theorem 3. Let $\mathcal{C}$ be a collection of countable sets such that $\mathcal{C}$ is countable. Then, $\mathcal{I} = \bigcup_{S \in \mathcal{C}} S$ is countable.

Proof sketch. We can write each element $c \in \mathcal{I}$ as (i, j) where i is the index of the set $S \in \mathcal{C}$ and j is the index of the element $x \in S$. Thus, $\mathcal{I}$ is isomorphic to some subset of $\mathbb{N} \times \mathbb{N}$. By Corollary 2 and Claim 4, $\mathbb{Q}$ is countable. □

Finally, we arrive at one of the most important theorems in countability.

Theorem 4 (Uncountability of reals)**.** $\mathbb{R}$ is uncountable.

Proof sketch. The proof is by Cantor's diagonal argument. □

Theorem 5 (Uncountability of powerset of naturals)**.** $\mathscr{P}(\mathbb{N})$ is uncountable.

Proof sketch. For any set $S \in \mathscr{P}(\mathbb{N})$, we can associate a unique real number r to S where $r = 0.\ldots$ where the i^{th} decimal place is 1 if the i^{th} natural number is in S and 0 otherwise. Hence, $\mathscr{P}(\mathbb{N})$ is isomorphic to $\mathbb{R}$. By Theorem 4, $\mathscr{P}(\mathbb{N})$ is uncountable. □

Next, let's look at alphabets, strings, and languages.

Theorem 6. Let Σ be some countable alphabet (or set). Σ^* is countable, where Σ^* denotes the set of all finite strings over Σ.

Theorem 7. If S be a countable set, then the set $\mathscr{P}_{\text{fin}}(S)$ of finite subsets of S is countable.

We have previously seen that every set of strings on a countable alphabet is countable. But what about the set of all languages?

Question 1. Let Σ be a countable alphabet. A language L over Σ is a subset of Σ^*. Is the collection of all languages over Σ countable? What if Σ is finite?

Motivations behind the Study of Logic

Professor Mathur's research interest is in the space of formal program verification. Formal program verification is the problem of determining if a program P meets a specification Φ; that is, $P \models \Phi$. This is a more complete way of verifying the correctness of programs compared to software engineering-style testing, but requires a background in mathematical logic.

Another motivation is in the realm of databases. We can model a query as a first-order logic formula, such as the formula $\phi \equiv \mathsf{Friends}(p_1, p_2)$ where the interpretation of $\mathsf{Friends}$ is $I(\mathsf{Friends}) = \{(p_1, p_2), (p_2, p_3), \ldots\}$ and the universe U is the set of all persons.

Yet another motivation is in the study of complexity theory, which discusses whether polynomial-time algorithms exist to solve a problem and what the best algorithm to solve a problem is. If we were able to encode a computational problem as a logic formulae, then determining whether the problem is solvable in polynomial time could be equivalent to determining the satisfiability of the formula.

Primer on Computability Theory

Computability Theory asks questions like "what is the complexity of solving a problem?", or more generally, "is the problem solvable?". We will study computability theory in the context of first-order logic (FOL), by modelling computational problems as formulae and determining if such formulae are satisfiable in polynomial time.

Here's an exercise: write a program P that takes

1. a program Q as input,
2. and an input I to Q as input.

such that P outputs "yes" if Q on input I prints "hello" as the first 5 characters, and "no" otherwise.

It turns out that it is impossible to write such a program.

Claim 7. *There is no program P that takes as input (1) a program Q and (2) an input I to Q such that P outputs "yes" if Q on input I prints "hello" as the first 5 characters, and "no" otherwise.*

Proof. Suppose, on the contrary, that there exists such a program P. Then, we can write a program P' which takes in a program Q and runs $P(Q, Q)$, then outputs "no" if $P(Q, Q)$ outputs "yes" else it outputs "no".

Now, suppose we execute $P'(P')$. If it outputs "no", then $P(P', P')$ outputs "yes", which means that $P'(P')$ outputs "hello". This is a contradiction. Otherwise, if $P'(P')$ outputs "hello", then $P(P', P')$ outputs "no", which means that $P'(P')$ does not output "hello". This is also a contradiction.

In either case, we arrive at a contradiction. Therefore, there is no such program P. □

The previous claim shows that certain computational problems that are unsolvable. Another well-known unsolvable problem is the Halting problem, which is the problem of determining whether a given program will terminate on a given input.

Turing Machines

We use Turing machines as our main model of computation. Input that is fed into a Turing machine can be modelled as strings over an alphabet. We will define some terms and notation before proceeding.

Definition 20 (Turing machine)**.** A *Turing machine* is a tuple $M = (Q, q_0, q_{\text{acc}}, q_{\text{rej}}, k, \delta, \Sigma)$ where

1. Q is a finite set of control states
2. Σ is the alphabet of tape symbols

3. $q_0 \in Q$ is the initial state
4. $q_{\text{acc}} \in Q$ is the accepting state
5. $q_{\text{rej}} \in Q$ is the rejecting state

Definition 21 (Configuration)**.** A *configuration* C of a Turing machine M is $C = (q, w_{\text{inp}} \uparrow w'_{\text{inp}}, w_{\text{WT}_1} \uparrow w'_{\text{WT}_1}, \ldots, w_{\text{WT}_k} \uparrow w'_{\text{WT}_k}, w_{\text{out}} \uparrow w'_{\text{out}})$ where w_x are finite strings over Σ representing the finite contents of the unbounded tape and $\uparrow$ refers to the pointer of the input tape, work tapes, and output tapes.

Definition 22 (Run/Computation)**.** A *run/computation* of Turing machine M on input $w \in \Sigma^*$ is $\pi = c_0, c_1, \ldots, c_m$ such that c_0 is the initial configuration and, for each i, c_{i+1} follows from c_i using δ.

Definition 23 (Accepting Run/Acceptance)**.** A run π is an *accepting run* if there is a configuration c in the run such that $c = (q_{\text{acc}}, \ldots)$ and q_{rej} is not reached before c. The input w is *accepted* by M iff the run π on M is an accepting run. Otherwise, w is rejected.

Definition 24 (Language of a Turing Machine)**.** The *language* of a Turing machine M is $L(M) = \{w \in \Sigma^* \mid w \text{ is accepted by } M\}$. A language $A \subseteq \Sigma^*$ is *recognized/accepted* by M if $A = L(M)$.

Definition 25 (Halting)**.** A Turing machine M *halts* on input w if there is a computation $\pi = c_0, c_1, \ldots, c_m$ such that $c_m = (q_{\text{acc}}, \ldots)$ or $c_m = (q_{\text{rej}}, \ldots)$. The run π is called a *halting run*.

For a list of objects $O_1, O_2, \ldots, O_k$, we will use $\langle O_1, O_2, \ldots, O_k \rangle$ to denote their binary encoding. In particular, for a Turing machine M, $\langle M \rangle$ is its encoding as a binary string. We may then define the language $L_{\text{Halt}} = \{\langle M, w \rangle \mid w \text{ on } M \text{ halts}\}$ Is there a Turing machine H such that $L(H) = L_{\text{Halt}}$?

The answer is yes: just simulate M on w. This Turing machine is known as a *universal Turing machine* since it can simulate the specification of an arbitrary Turing machine on arbitrary input.

Recursive and Recursively Enumerable Languages

Recall that there are 3 possible outcomes when a Turing machine M runs on an input string w — M may halt and accept w, M may halt and reject w, or M may not halt on w. Depending on how a Turing machine behaves, we can define two different classes of problems solvable on a Turing machine.

Definition 26 (Recursively Enumerable)**.** A language A is *recursively enumerable/semi-decidable* if there is a Turing machine M such that $A = L(M)$. We denote the set of all recursively enumerable languages as RE.

Example. L_{Halt} is recursively enumerable. That is, $L_{\text{Halt}} \in \mathsf{RE}$.

Definition 27 (Recursive/Decidable)**.** A language A is *recursive/decidable* if there is a Turing machine M that halts on *all* inputs and $A = L(M)$. We denote the set of all recursive languages as REC.

As an example, consider the language $L_{\text{Sorted}} = \{\langle l \rangle \mid l \text{ is a sorted list}\}$.

Example. $L_{\text{Sorted}} \in \mathsf{REC}$ but $L_{\text{Halt}} \notin \mathsf{REC}$.

Observe that a problem that is recursive is solvable by an algorithm that always halts. Thus, by definition, recursive languages are also recursively enumerable. This observation is equivalent to the following lemma:

Lemma 3 (Recursive Implies Recursively Enumerable)**.** *REC* $\subseteq$ *RE*.

We also define the complement of a language.

Definition 28 (Complement of a Language)**.** The complement of a language L is $\overline{L} = \Sigma^* \setminus L$.

Theorem 8 (Complement of Recursive is Recursive)**.** If $L \in \mathsf{REC}$, then $\overline{L} \in \mathsf{REC}$.

Proof sketch. Take the Turing machine M that accepts L and let M' be the Turing machine M with the only difference being that the accepting and rejecting states are swapped. Then, $\overline{L} = L(M')$ and M' halts on every input. Thus, $\overline{L} \in \mathsf{REC}$. □

The following theorem is a useful way to prove that a problem is recursive.

Theorem 9. L is recursive iff L and $\overline{L}$ are recursively enumerable. That is, $L \in \mathsf{REC}$ iff $L \in \mathsf{RE}$ and $\overline{L} \in \mathsf{RE}$.

Proof sketch. The (easy) forward direction uses Lemma 3.

The reverse direction uses a technique called *dovetailing*. Suppose that L and $\overline{L}$ are recognized by M and $\overline{M}$ respectively. Then, construct M' by running both M and $\overline{M}$ simultaneously on an input w, and accept if M accepts or reject if $\overline{M}$ accepts. Since w belongs to either L or $\overline{L}$, M' halts on all inputs and $L = L(M')$. Thus, $L \in \mathsf{REC}$. □

Not every decision problem is recursively enumerable.

Theorem 10 (Language outside RE)**.** There is a language L that is not recursively enumerable. That is, $L \notin \mathsf{RE}$.

Proof. Since $L_{\text{Halt}} \notin \mathsf{REC}$ but $L_{\text{Halt}} \in \mathsf{RE}$, by Theorem 9, $\overline{L_{\text{Halt}}} \notin \mathsf{RE}$.

Alternatively, note that there are uncountably many languages yet countably many Turing machines. □

Next, consider the language $K = \{\langle M \rangle \mid \langle M \rangle \in L(M)\}$. Using Cantor's diagonalization technique, we can establish that the complement $\overline{K}$ is not recursively enumerable.

Theorem 11. The language $\overline{K} = \{\langle M \rangle \mid \langle M \rangle \notin L(M)\}$ is not recursively enumerable. That is, $\overline{K} \notin \mathsf{RE}$.

Proof. Suppose to the contrary that $\overline{K} = L(M)$ for some Turing machine M. If $\langle M \rangle \in \overline{K}$, then $\langle M \rangle \notin L(M) = \overline{K}$, which is a contradiction. Otherwise, if $\langle M \rangle \notin \overline{K}$, then $\langle M \rangle \in L(M) = \overline{K}$, which is also a contradiction. Therefore, $\overline{K} \neq L(M)$ for any Turing machine M and so $\overline{K} \notin \mathsf{RE}$. □