CS 4269/5469– Fundamentals of Logic In Computer Science

SEMESTER II, 2022-2023

Overall Notes

Primer on Countability

Definition 1 (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 2 (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 1 (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 1, 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*.

Remark. *Some examples of Claim 1 are the set of real numbers $\mathbb{R}$ and the powerset of the natural numbers $\mathscr{P}(\mathbb{N})$.*

Next, we will state a straightforward claim.

Claim 2. *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 3. *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 $s \in S'$. $\square$

Remark. *As a result of Claim 3, some other examples of countable sets are: the set of even numbers, the set of odd numbers, and the set of 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 4. *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 5. *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 5 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 5 yields the following corollaries:

Corollary 1. $\mathbb{N} \times \mathbb{N}$ *is countable.*

Corollary 2. $\mathbb{Q}$ *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 1 and Claim 3, $\mathbb{Q}$ is countable. $\square$

Corollary 3. $\mathbb{Z}$ *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 1 and Claim 3, $\mathbb{Q}$ is countable. $\square$

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

Remark. *An example of Theorem 2 is the powerset of the natural numbers $\mathscr{P}(\mathbb{N})$.*

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 1 and Claim 3, $\mathbb{Q}$ is countable. $\square$

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. $\square$

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. $\square$

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, $P \vDash \Phi$. 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 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 6. *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. $\square$

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 3 (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. $q_{\mathrm{acc}} \in Q$ is the accepting state

5. $q_{\mathrm{rej}} \in Q$ is the rejecting state

Definition 4 (Configuration). A *configuration* C of a Turing machine M is $C = (q, w_{\mathrm{inp}} \uparrow w'_{\mathrm{inp}}, w_{\mathrm{WT}_1} \uparrow w'_{\mathrm{WT}_1}, \ldots, w_{\mathrm{WT}_k} \uparrow w'_{\mathrm{WT}_k}, w_{\mathrm{out}} \uparrow w'_{\mathrm{out}})$ 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 5 (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 6 (Accepting Run/Acceptance). A run π is an *accepting run* if there is a configuration c in the run such that $c = (q_{\mathrm{acc}}, \ldots)$ 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 7 (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 8 (Halting). A Turing machine M *halts* on input w if there is a computation $\pi = c_0, c_1, \ldots, c_m$ such that $c_m = (q_{\mathrm{acc}}, \ldots)$ or $c_m = (q_{\mathrm{rej}}, \ldots)$. 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 $L_{\mathrm{Halt}} = \{\langle M, w \rangle \mid w \text{ on } M \text{ halts}\}$ Is there a Turing machine H such that $L(H) = L_{\mathrm{Halt}}$?

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 9 (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.

Remark. $L_{Halt} \in RE$.

Definition 10 (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 $L_{\text{Sorted}} = \{\langle l \rangle \mid l \text{ is a sorted list}\}$.

Remark. $L_{Sorted} \in REC$ *but* $L_{Halt} \notin REC$.

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 1 (Recursive Implies Recursively Enumerable). $REC \subseteq RE$.

We also define the complement of a language.

Definition 11 (Complement of a Language). The complement of a language L is $\overline{L} = \Sigma^* \setminus L$.

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

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, $\overline{L} \in \text{REC}$. □

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, $L \in \text{REC}$ iff $L \in \text{RE}$ and $\overline{L} \in \text{RE}$.

Proof sketch. The (easy) forward direction uses Lemma 1.

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, $L \in \text{REC}$. □

Not every decision problem is recursively enumerable.

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

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

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 $\overline{K} \notin \mathsf{RE}$. $\square$