FLAT recap

1 Introduction

In this course we learnt about languages, both finite and infinite. Given an alphabet Σ, a language L is a subset of strings where each element belongs to Σ - in other words $L \subseteq \Sigma^*$. We primarily learnt three ways in which some (possibly infinite) languages can have finite representations:

1. Finite automata/ regular expressions
2. Pushdown Automata/ Context free grammars
3. Turing Machine / Unrestricted grammars

We also learnt that not every language can have a finite encoding. In fact, for each of the above 3 representations we saw examples of languages that could not have that representation.

2 Finite Automata/regular expressions

Languages that are accepted by Finite Automata are called regular languages. A language is accepted by a Finite Automaton if and only if it can be expressed as a regular expression. We learnt an algorithm to convert an automaton to a regular expression and vice versa. Thus, in order to show that a language is regular we learnt the following tools:

1. Design a deterministic/non-deterministic Finite Automaton for the language. For every non-deterministic finite automaton we can design a deterministic finite automaton (subset construction algorithm).
2. Find out a regular expression for the language.
3. Use closure properties. For example, if the given language is the union of two regular languages, then since regular languages are closed under union we can derive that the given language is also regular. We learnt a set of operations under which regular languages are closed.
4. If L is a finite language, it has finitely many strings. For each string we can design a finite automaton accepting only that string - as regular languages are closed under finite unions, we can take the union of these finite automata.

5. The pumping Lemma CANNOT be used to show that a language is regular!

Not all languages are regular. Tools for showing that a language is not regular:

1. Pumping Lemma. The idea behind using the pumping lemma to show that a language L is not regular is to assume by contradiction that a deterministic finite automaton N with k states exists and start with a string s of length much larger than k (L cannot be finite as otherwise it is definitely regular). It must be the case that at least one state p in N will be repeated in the path in N that accepts s. Let v be the substring of s that appears between consecutive occurrences of the state p on the path of acceptance for s. Even if we create a new string s' by either throwing v out, or by pumping in multiple copies of v, N should accept the new string s'. So if there is a pumping constant i such that pumping v i times creates a new string s' that does not belong to the language L, then we have arrived at a contradiction.
   The strategy to show that a language L is not regular using the Pumping Lemma is an adversarial game.
   (a) The adversary gives an integer $k \geq 0$,
   (b) you pick an n-length string $s = xyz$ such that $s \in L$ and $|y| \geq k$,
   (c) The adversary gives you substrings u, v, w such that $y = uvw$ and $v \neq \epsilon$
   (d) You pick $i \geq 0$
   (e) You win if $xuv^iwz \notin L$.

2. We also learnt about ultimate periodicity as a tool for showing that a language is not regular.

3. Closure properties can be used here as well. For example, suppose you know that the given language and another regular language in union forms a language that is known not to be regular, then the given language cannot be regular - otherwise the language created in union should also be regular.

3 Pushdown Automata/Context-free grammars

Languages that are accepted by nondeterministic Pushdown Automata are called context=free languages. A language is accepted by a nondeterministic Pushdown Automaton if and only if it has a context-free grammar. We learnt an algorithm to convert a nondeterministic pushdown automaton to a context-free grammar and vice versa.

A regular language is also a context-free language. We learnt that a regular language can be expressed as a right-linear/left-linear grammer. So a regular language is a special case of context-free languages.

Thus, in order to show that a language is context-free we learnt the following tools:

1. Design a non-deterministic Pushdown Automaton for the language.
   A strict subset of context-free languages are accepted by deterministic pushdown automata. We saw examples of context-free languages that do not have deterministic pushdown automata.

2. Find out a context-free grammar for the language.

3. Use closure properties. For example, if the given language is the union of two context-free languages, then since context-free languages are closed under union we can derive that the given language is also context-free. We learnt a set of operations under which context-free languages are closed.

4. The pumping Lemma for CFLs CANNOT be used to show that a language is context-free!

Not all languages are context-free. Tools for showing that a language is not context-free:

1. Pumping Lemma for CFLs. Briefly, the idea behind this pumping lemma is to show that if there are $< logk$ nonterminals in the context-free grammar of a language, then in order to derive a k-length string s we are forced to use at least one non-terminal twice (could be in different steps of the the derivation). Then, in the later appearance of the non-terminal, we can repeat the intermediate derivation pattern or delete the intermediate derivation pattern completely. The formal argument can be understood using parse trees/derivation trees.
   The strategy to show that a language L is not context-free using the Pumping Lemma for CFLs is an adversarial game.
   (a) The adversary gives an integer $k \geq 0$,
   (b) you pick a string s such that $s \in L$ and $|s| \geq k$,
   (c) The adversary gives you substrings u, v, w, x, y such that $s = uvwxy$, $|vx| > 0$, and $|vwx| \leq k$,
   (d) You pick $i \geq 0$
   (e) You win if $uv^iwx^iz \notin L$.

2. Closure properties can be used here as well. For example, suppose you know that the given language and another context-free language in union forms a language that is known not to be context-free, then the given language cannot be context-free - otherwise the language created in union should also be context-free.

4 Turing Machines/unrestricted grammars

Languages that are accepted by Turing machines are called semidecidable languages/recursively enumerable sets. A language is accepted by a Turing machine if and only if it has an unrestricted grammar. Towards the last part of the course, we learnt an algorithm to convert a Turing Machine to an unrestricted grammar and vice versa.

Context-free languages are also recursively enumerable, because CFGs are a special case of unrestricted grammar. Similarly, regular languages with their linear grammars are recursively enumerable.

In order to show that a language is context-free we learnt the following tools:

1. If we want to show that a language is recursively enumerable then we design a non-deterministic/deterministic Turing machine for the language. Non-determinism does not add any power to Turing machines, although the number of steps required by a nondeterministic TM can be exponentially less than that of a corresponding deterministic TM.
   A strict subset of recursively enumerable languages are recursive languages/decidable languages. These are accepted by Turing machines with the following property: given any input string either it is accepted by the TM or it is rejected by the TM. Thus the TM halts on each input string; it does not loop on any input string.

2. For recursively enumerable languages, find out an unrestricted grammar for the language.

3. Use closure properties. For example, if the given language is the union of two recursively enumerable languages, then since recursively enumerable languages are closed under union we can derive that the given language is also recursively enumerable. We learnt a set of operations under which recursively enumerable/recursive languages are closed.

Not all languages are recursive or recursively enumerable. Tools for showing that a language is not recursive/ recursively enumerable:

1. Diagonalization technique, similar to Cantor's Diagonalization argument.

2. Reductions. Suppose we are given two sets $A \subseteq \Sigma^*$ and $B \subseteq \Sigma^*$. A reduction is a (possibly many-one) computable function $\sigma : \Sigma^* \to \Delta^*$ such that for all $x \in \Sigma^*$, $x \in A \iff \sigma(x) \in B$.
   Things to remember: a function f is said to be computable if there is a TM that takes an input x and at the end of computation writes down $f(x)$ on its work-tape.
   Thus, We are essentially using a Turing machine that converts a string x in Σ into a string $\sigma(x) \in \Sigma$ such that $x \in A$ if and only if $\sigma(x) \in B$.
   Now, if B is a recursively enumerable set then so is A. On the other hand, if A is not a r.e set then B cannot be.
   Similarly, if B is a recursive set, then so is A. On the other hand, if A is not recursive then B cannot be.
   To show that a language B is not recursive, give a reduction from a language A like the Halting Problem HP, which is known to not be recursive. IF you want to show that B is not r.e, give a reduction from a language A like $\overline{HP}$, which is known to be not r.e.

3. Rice's Theorems.