Professor Parikh
Professor Pedersen
Luka Nikabadze

Chapter 1 Regular Languages

Computational model describes how an output of a a mathematical function is computed given an input. The simplest models that we have called finite state machie or finite automation. A finite automaton (FA) is a simple idealized machine used to recognize patterns within input taken from some character set (or alphabet) C. The job of an FA is to accept or reject an input depending on whether the pattern defined by the FA occurs in the input. A finite automaton has a 5-tuple (Q, Σ, δ, q0, F), where

1. Q is a finite set called the states,

2. Σ is a finite set called the alphabet,

3. $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,

4. $q0 \in Q$ is the start state, and

5. $F \subseteq Q$ is the set of accept states.

If the machine accepts its input if it ends up in an accept state. We say that M recognizes language A if A = {w| M accepts w}. so a language is called a regular language if some finite automation recognizes it.To develop techniques for desgning automata to recognize if language is regular or not we have The regular operations union, concatenation, and star as follows:

- **Union: $A \cup B = \{x| x \in A \text{ or } x \in B\}$.**
- **Concatenation: $A \circ B = \{xy| x \in A \text{ and } y \in B\}$.**
- **Star: $A* = \{x1x2 ...xk| k \geq 0 \text{ and each } xi \in A\}$.**

There can be nondeterministic dn deterministic finite automata. deterministic finite automaton, if each of its transitions is uniquely determined by its source state and input symbol, and reading an input symbol is required for each state transition. deterministic automaton is a concept of automata theory in which the outcome of a transition from one state to another is determined by the input.

the pumping lemma for regular languages is a lemma that describes an essential property of all regular languages. Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, s = xyz, satisfying the following conditions:

1. For each $i \geq 0$, $xyiz \in A$, **2. $|y| > 0$, and** **3. $|xy| \leq p$.**

Professor Parikh
Professor Pedersen
Luka Nikabadze

Chapter 2 Context Free Grammar

A formal grammar is considered "context free" when its production rules can be applied regardless of the context of a nonterminal. No matter which symbols surround it, the single nonterminal on the left hand side can always be replaced by the right hand side. Let's formalize our notion of a context-free grammar (CFG).

A *context-free grammar* is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the *variables*,

2. Σ is a finite set, disjoint from V , called the *terminals*,

3. R is a finite set of *rules*, with each rule being a variable and a string of variables and terminals, and

4. S ∈ V is the start variable.

A string w is derived ambiguously in context-free grammar G if it has two or more different leftmost derivations. Grammar G is ambiguous if it generates some string ambiguously. A context-free grammar is in Chomsky normal form if every rule where

A → BC

A → a

a is any terminal and A, B, and C are any variables—except that B and C may not be the start variable. In addition, we permit the rule S → ε, where S is the start variable. Like from the previous chapter pumping lemma was showed that certain languages are not regular. Here is this case we also have same concept for context free languages. It states that every context-free language has a special value called the pumping length such that all longer strings in the language can be "pumped." This time the meaning of pumped is a bit more complex. It means that the string can be divided into five parts so that the second and the fourth parts may be repeated together any number of times and the resulting string still remains in the language. The deterministic pushdown automata accepts the deterministic context-free languages, a proper subset of context-free languages.

A deterministic pushdown automaton is a 6-tuple (Q, Σ, Γ, δ, q0, F),where Q, Σ, Γ, and F are all finite sets, **and 1. Q is the set of states, 2. Σ is the input alphabet, 3. Γ is the stack alphabet, 4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow (Q \times \Gamma_\varepsilon) \cup \{\emptyset\}$ is the transition function, 5. $q_0 \in Q$ is the start state, and 6. $F \subseteq Q$ is the set of accept states.**

Professor Parikh
Professor Pedersen
Luka Nikabadze

Chapter 3 The church Turing thesis

A Turing machine is a mathematical model of computation that defines an abstract machine, which manipulates symbols on a strip of tape according to a table of rules. A Turing machine can do everything that a real computer can do. To be more clear the main differences between Turing machine and finite automata is that A Turing machine can both write on the tape and read from it. The read–write head can move both to the left and to the right. The tape is infinite.The special states for rejecting and accepting take effect immediately.

A Turing machine is a 7-tuple, (Q, Σ, Γ, δ, q0, qaccept, qreject), where
Q, Σ, Γ are all finite sets and

1. Q is the set of states,

2. Σ is the input alphabet not containing the blank symbol ␣,

3. Γ is the tape alphabet, where ␣ ∈ Γ and Σ ⊆ Γ,

4. δ : Q × Γ−→Q × Γ × {L, R} is the transition function,

5. q0 ∈ Q is the start state,

6. Q accept ∈ Q is the accept state, and

7. Q reject ∈ Q is the reject state, where q reject ≠ q accept.

nondeterministic Turing machine is a theoretical model of computation whose governing rules specify more than one possible action when in some given situations. We can simulate any nondeterministic TM N with a deterministic TM D. The idea behind the simulation is to have D try all possible branches of N's nondeterministic computation. If D ever finds the accept state on one of these branches, D accepts. Otherwise, D's simulation will not terminate. an enumerator is a Turing machine with an attached printer. The Turing machine can use that printer as an output device to print strings. A language is Turing-recognizable if and only if some enumerator enumerates it.
First we show that if we have an enumerator E that enumerates a language A, a TM M recognizes A. The TM M works in the following way. M = "On input w:

1. Run E. Every time that E outputs a string, compare it with w.

2. If w ever appears in the output of E, accept"

Professor Parikh
Professor Pedersen
Luka Nikabadze

Chapter 4 Decidability

Acc A language for which membership can be decided by an algorithm that halts on all inputs in a finite number of steps --- equivalently, can be recognized by a Turing machine that halts for all inputs is known as decidable language. The way we prove if language is decidable is that we simply need to present a TM M that decides A DFA.
M = “On input ⟨B, w⟩, where B is a DFA and w is a string:

1. Simulate B on input w.

2. If the simulation ends in an accept state, accept. If it ends in a

nonaccepting state, reject.”
we can also prove similar theorem for nondeterministic finite automata. So A NFA is a decidable language.

N = “On input ⟨B, w⟩, where B is an NFA and w is a string:
1. Convert NFA B to an equivalent DFA C
2. Run TM M from Theorem 4.1 on input ⟨C, w⟩.
3. If M accepts, accept; otherwise, reject.”
Every context free language can be decidable. Let G be a CFG for A and design a TM M G that decides A. we build a copy of G into MG. and it works as follows.
MG = “On input w: **1. Run TM S on input ⟨G, w⟩.**
2. If this machine accepts, accept; if it rejects, reject.”
Now we turn to our first theorem that establishes the undecidability of a specific language: the problem of determining whether a Turing machine accepts a given input string. We call it ATM by analogy with ADFA and ACFG. But, whereas ADFA and ACFG were decidable, ATM is not. Let ATM = {⟨M,w⟩| M is a TM and M accepts w}.
A TM is undecidable. The following Turing machine U recognizes ATM.
U = “On input ⟨M,w⟩, where M is a TM and w is a string:
1. Simulate M on input w.
2. If M ever enters its accept state, accept; if M ever enters its
reject state, reject.”

Professor Parikh
Professor Pedersen
Luka Nikabadze

Chapter 5 Reducibility

Problems that are computationally unsolvable it is called reducibility. A reduction is a way to convert one problem to another problem in such a way that a solution to the second problem can be used to solve the first problem. Let's consider a related problem, HALT TM, the problem of determining whether a Turing machine halts (by accepting or rejecting) on a given input. This problem is widely known as the halting problem. We use the undecidability of ATM to prove the undecidability of the halting problem by reducing ATM to HALT TM.

Let HALT TM = {⟨M,w⟩| M is a TM and M halts on input w}.

Let's assume for the purpose of obtaining a contradiction that TM R decides HALT TM. We construct TM S to decide ATM, with S operating as follows.

S = "On input ⟨M,w⟩, an encoding of a TM M and a string w:

1. Run TM R on input ⟨M,w⟩.

2. If R rejects, reject.

3. If R accepts, simulate M on w until it halts.

4. If M has accepted, accept; if M has rejected, reject."

we can also prove similar theorem so E tm is undecidable. We assume that ETM is decidable and then show that ATM is decidable—a contradiction. Let R be a TM that decides ETM. We use R to construct TM S that decides ATM. How will S work when it receives input ⟨M,w⟩? Let's write the modified machine described in the proof idea using our standard notation. We call it M1.

M1 = "On input x:

1. If x ≠ w, reject.

2. If x = w, run M on input w and accept if M does."

we can also prove similar theorem so EQ tm tm is undecidable. We let TM R decide EQTM and construct TM S to decide ETM as follows.

S = "On input ⟨M⟩, where M is a TM: 1. Run R on input ⟨M,M1⟩, where M1 is a TM that rejects all inputs. 2. If R accepts, accept; if R rejects, reject." If R decides EQTM, S decides ETM. But ETM is undecidable