Theory of Computation

Lecture delivered by Prof. Stephen Kobourov

Shuo Yang

October 13, 2014

Last updated: October 13, 2014

1 Introduction

In this class, we explore into different types of *computational model* which is an idealized computer. We start with finite automata.

2 Regular Languages

Finite Automata are good models for computers with an extremely limited amount of memory.

2.1 DFA

Formal definition of a DFA

A DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the *states*,
2. Σ is a finite set called the *alphabet*,
3. δ: $Q \times \Sigma \rightarrow Q$
4. $q_0 \in Q$ is the *transition function*
5. $F \subseteq Q$ is the *set of accept states*

2.2 Intuition behind the pumping lemma for regular languages

No matter how tricky a regular language seems to be, there must be some repetitive pattern inside the language which can be tracked by a finite number of states.

3 The Church-Turing Thesis

3.1 Turing Machine

A *Turing Machine* is a much more accurate model of a general purpose computer. It can do everything that a real computer can do. A TM can end up in an *accept state, reject state, or an infinite loop.*

Formal definition of a Turing Machine

A TM is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_accept, q_reject)$, where

1. Q is the set of states,
2. Σ is the input alphabet not containing the blank symbol $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, and $q_{accept} \neq q_{reject}$.

The definition tells us that as a TM computes, changes occurs in the current state, the current tape contents, and the current head location. A setting of these three items is called a ***configuration*** of the Turing Machine. Say that configuration C_1 ***yields*** configuration C_2 if the Turing Machine can legally go from C_1 to C_2 in a single step.

Three special configurations:

1. ***start configuration*** of M on input w is the configuration q_0w, which indicates that the machine is in the start state q_0 with its head at the leftmost position on the tape.
2. ***accepting configuration*** the state of the configuration is q_{accept}.
3. ***rejecting configuration*** the state of the configuration is q_{reject}.

Both *accepting configuration* and *rejecting configuration* are ***halting configurations*** and do not yield further configurations. (*If a machine does not accept a string, it doesn't mean it rejects, it could either reject or enter an infinite loop.*)

The collection of strings that a Turing machine M accepts is the language of M, or the language recognized by M, denoted $L(M)$.

Definition
Call a language ***Turing-recognizable*** (a.k.a *recursively enumerable language*) if some Turing machine recognize it. (such Turing machine may halt on inputs, or loop forever.)

Definition
Call a language ***Turing-decidable*** (a.k.a *recursive language*) or simply ***decidable*** if some Turing machine decides it. (such Turing machine always halts on all inputs.)

Example
A Turing machine that decides the language $\{w\#w | w \in \{0, 1\}^*\}$.
M = "On input string w:

1. scan from the leftmost position on the tape, cross off the first unmarked symbol. Then move the head all the way to the right to the corresponding position on the other side of the # to check whether these two positions contain the same symbol. If they do not, or if no # is found, *reject*. If they do, cross off the matched symbol and move the head all the way back to the leftmost position.
2. repeat the previous step until all the symbols to the left of # have been crossed off, check for any remaining symbols to the right of the #, if any symbols remain, *reject*, otherwise, *accept*."

Example
A Turing machine that decides the language $\{0^{2^n} | n \geq 0\}$.
M = "On input string w:

1. From left to right, cross off every other 0, if the tape contains a single 0, *accept.* Otherwise, if the number of $0's$ is odd but not 1, *reject.*
2. Return to the left-hand end of the tape.
3. Go to stage 1."

Example
A Turing machine that decides the language $\{a^i b^j c^k | i \times j = k$ and $i, j, k \geq 1\}$.
M = "On input string w:

1. Scan the input from left to right to check if the input is in the form of $a^+b^+c^+$ and reject if it isn't.
2. Return the head to the left-hand end of the tape.
3. Cross off an a and scan to the right until a b is found. Shuttle between the $b's$ and the $c's$, crossing off one of each until all $b's$ are gone. If all $c's$ have been crossed off and some $b's$ remain, *reject.*
4. Restore the crossed off $b's$ and repeat stage 3 if there is another a to cross off. If all the $a's$ have been crossed off, check whether all the $c's$ also have been crossed off. If yes, *accept*, otherwise, *reject.*"

Example
A Turing machine that decides the language
$\{\#x_1\#x_2\# \cdots \#x_l |$ each $x_i \in \{0,1\}^*$ and $x_i \neq x_j$ for each $i \neq j\}$.
M = "On input string w:

1. Place a mark on top of the leftmost symbol. If that symbol was a blank, *accept.* If it was a #, continue with the next stage. Otherwise, *reject.*
2. Scan right to the next # and place a second mark on top of it. If no # is found before a blank symbol, only x_1 was present, *accept.* Otherwise, continue with the next stage.
3. By zigzagging, compare the two strings to the right of the marked #s. If they are equal, *reject.* Otherwise, continue with the next stage.
4. Move rightmost of the two marks to the next # to the right. If no # is encountered before a blank symbol, move the leftmost mark to the next # to the right and rightmost mark to the # after that. If no # is available for the rightmost mark, all the strings have been compared, so *accept.*
5. Go to stage 3."

3.2 Variants of Turing Machine

The original Turing machine model and its reasonable variants all have the same power - ***they recognize the same class of languages***. This invariance to certain changes is called ***robustness***. Both FAs and PDAs are somewhat robust models, but TMs have an as astonishing degree of robustness. The idea of proving two TMs are equivalent in power is simply to show that we can simulate one by another.

4 Midterm Summary

4.1 RL, CFL

$\{w|$w has an equal number of 0s and 1s$\}$ is not a regular language, but is a CFL.
$\{1^{n^2} \mid n \geq 0\}$ is not a regular language. Is it CFL? No. Is it context sensitive language (CSL)?
$\{0^n1^n \mid n \geq 0\}$ is both CFL and DCFL.
$\{a^ib^jc^k \mid i, j, k \geq 0$ and $i = j$ or $i = k\}$ and $\{ww^R|w \in \{0, 1\}^*\}$ are CFLs, but not DCFLs.
$\{ww|w \in \{0, 1\}^*\}$ is not CFL.
$\{a^ib^jc^k \mid 0 \leq i \leq j \leq k\}$ is not a CFL.
$\{a^nb^nc^n \mid 0 \leq n\}$ is not a CFL, but it is a CSL.
$\{1^p \mid p$ is a prime number$\}$ is not a CFL, but is a CSL.

The class of DCFLs is closed under complement, but **not** closed under union, concatenation and Kleene-star.
If a language has a DPDA to recognize it, then it has an unambiguous grammar.
The other way doesn't work! For example, $\{ww^R|w \in \{0, 1\}^*\}$ has unambiguous CFG, but it is not a DCFL.

4.2 Decidability