CIS 511: Theory of Computation **Jan 19, 2017**

Lecture 3: Regular Language Wrap-up and Intro to Turing Machines

Professor Sampath Kannan *Zach Schutzman*

NB: *These notes are from CIS511 at Penn. The course followed Michael Sipser's Introduction to the Theory of Computation (3ed) text.*

The rest of Regular Languages

Recall, regular languages are those recognized by DFAs, NFAs, and described by regular expressions

Are there languages that are not regular? Trivially, yes. The set of all languages is uncountable, but the set of all regular languages is countable.

How can we determine that a language is *not* regular?

Theorem 3.1 ***The Pumping Lemma*** *(for regular languages): If L is a regular language, then there exists a constant p, called its* **pumping length**, *such that if $x \in L$ and $|x| \geq p$, then x can be written as $x = uvw$, where v is non-empty, $|uv| \leq p$, and $uv^iw \in L$ for any $i \in \mathbb{N}$.*

Proof: *Suppose L is regular. Then there exists a DFA M that recognizes L. Say M has p states ($|Q| = p$). Suppose L contains some string x, $|x| > p$. Then by the Pigeonhole Principle, the path of x must pass through at least one state at least twice. Equivalently, the path of x contains some cycle.*

Let's think of x as broken into 3 parts: uvw, where u is the portion before the cycle, v is the cycle, and w is the portion after the cycle. We will allow u or w to be empty, but v must be non-empty. Note that $x \in L$ implies $uw \in L$, and more generally that $uv^iw \in L$, for any $i \in \mathbb{N}$.

We also note that $|uv| \leq p$, that is there will always be a cycle before processing p characters.

■

Corollary 3.2 *A language is not regular if there does not exist any way to break a string of length greater than p into a satisfactory uvw.*

Example:

Claim 3.3 *The language $L = \{0^n1^n | n \in \mathbb{N}\}$ is not regular.*

Proof:

Suppose, for the sake of contradiction, that L is regular with pumping length p. Consider the string 0^p1^p. Since we have $|uv| \leq p$, v is of the form 0^k, for some $1 \leq k \leq p$. The Pumping lemma asserts that uv^iw is in L for all $i \in \mathbb{N}$. But the string uv^2w has more 0s than 1s, and is not in L. This is a contradiction, therefore L is not a regular language.

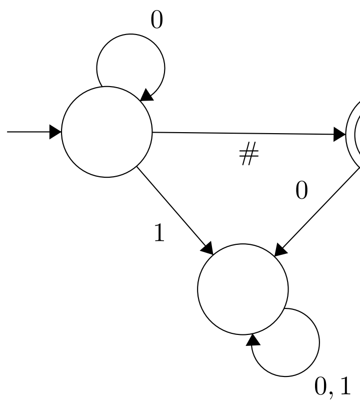
■

Example:

Claim 3.4 *The language $L = \{x\#y||x| = |y|\}$ is not regular.*

Proof:

The class of regular languages is closed under intersection. Suppose, for the sake of contradiction, that L is regular. Let the language $K = \{0^*\#1^*\}$. K is regular, as it is recognized by the following machine:



Let's consider $K \cap L$. This language is $K \cap L = \{0^n\#1^n|n \in \mathbb{N}\}$. By a slight adjustment to the previous proof, we can say that $K \cap L$ is not regular. This contradicts the assumption that both K and L are regular, and since we know that K is regular, it must be the case that L is not.

■

Corollary 3.5 DFAs can't count!

Turing Machines

We are moving from the simplest model of computation to the 'most'. We can think of a DFA as receiving an input on a tape, reading one character at a time, and only moving forward, and changing states according to the currently read character and the current state, via δ.

We can identify three principal limitations:

1. *The head doesn't pass the end of the tape*
2. *The tape is read-only*
3. *The head only moves forward*

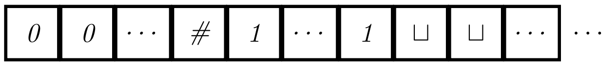
For a Turing machine, we lift all three of these limitations. The head can move left and right, it can read and write, and it can move beyond the input.

We are going to expand the tape alphabet Γ to be a proper superset of Σ, and also containing a blank symbol $\sqcup$.

Claim 3.6 *Our language $L = \{0^n\#1^n | n \in \mathbb{N}\}$ can be recognized by a Turing Machine.*

Proof:

Consider an example input below. First, we can scan the string to validate it is of the correct form, if not, reject. Assume that it is.



The following algorithm determines if the string is in L:

Replace the first zero on the tape with a special marker symbol.

Traverse the tape and replace the first one with a marker symbol.

Repeat until you either run out of zeros or ones.

If all symbols are marked, accept, otherwise, reject.

■

Definition 3.7 *A* ***Turing Machine*** *is a 7-tuple $(Q, q_0, \Sigma, \Gamma, \delta, q_a, q_r)$.*

Q is the set of states

q_0 is the start state

Σ is the input alphabet

Γ is the tape alphabet, a proper superset of Σ

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function.

q_a is the accept state

q_r is the reject state.

The transition function maps the current state and the currently read character to the next state, the character to write on that cell, and a LEFT or RIGHT move. The machine halts when it reaches q_a or q_r, by definition.

Definition 3.8 *Given a Turing machine M, the language* ***recognized*** *by M is the set of all strings that cause M to move to q_a.*

If M recognizes L and $x \notin L$, there are two possibilities:

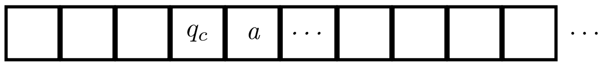
1. *M reaches q_r*
2. *M computes infinitely and never reaches q_a or q_r. That is, M 'loops forever'.*

In both of these cases, we say that M doesn't recognize x, but in the second case, how do we know if M really won't halt on x, rather than just taking a really long time?

Definition 3.9 *A Turing machine M **decides** a language L if for all $x \in L$, M reaches q_a and for all $x \notin L$, M reaches q_r. The set of Turing decidable languages is a subset of the Turing recognizable languages.*

Definition 3.10 *The **current configuration** of a Turing machine M is the current state, the tape contents, and the head position at some point in time.*

The notation for this is:



to denote that we are in state q_c and the head is over a.

Example: *Let's construct a Turing machine to recognize the language of palindromes, $L = \{w\#w^R | w = (w^R)^R\}$.*