Lecture Notes for CS 509
Foundations of Computer Science
Lecture 2

Lecturer: Joe Kilian
Scribe: Brian Thompson

1 Topics to discuss:

- How do we show languages are regular?
- How do we show languages are not regular?
- Addressing problems beyond determining membership in a language
- Evaluating efficiency (NFA to DFA, DFA to RegExp, etc.)
- Extending our models of finite state machines
- Context-Free Languages (won't spend much time, just read the book)

2 Questions about Regular Languages

Lemma 2.1. *L_1, L_2 regular $\Longrightarrow L_1 \cap L_2$ regular.*

- Proof 1: Since we know regular languages are closed under Union and Complement, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ must also be regular.
- Proof 2: The cross-product idea. Let M_1, M_2 be DFAs for L_1, L_2. Keep track of where you are in M_1 and M_2 simultaneously, and accept a string iff both machines would accept it. Since each machine only requires a constant amount of memory, keeping track of both is still constant size.

The ways in which Ronald Reagan benefited humanity may be far from numerous, but he did leave us with one great phrase: *"trust but verify"*. This is a good way of thinking about non-determinism. Assume that every possible next step will happen, and then when you find out what actually happened, scrap the bad guesses.

What if a DFA is allowed to view not just the current symbol in the string, but also the symbols immediately before and after? A normal DFA is just as powerful!
Let T be one of these triple-view DFAs.

- Proof 1: Make a new NFA M that stores the previous and current symbols in its state, and have it predict the next symbol non-deterministically (by the so-called Reagan's Rule).
- Proof 2: Make a new DFA M with a different alphabet - each triplet of symbols in T becomes one new symbol in M. Then just make sure that the symbols follow correctly.

Regular expressions are clearly closed under Reverse (leave Unions and Kleene Stars as is, and reverse all Concatenations), but what is the intuition for DFAs?

- Think of tracking the reverse DFA non-deterministically. You only need to keep track of the last state.

3 Nerode Equivalence Classes

Definition. $x_1 \sim_L x_2 \quad \text{iff} \quad (\forall y) \;\; x_1 y \in L \Leftrightarrow x_2 y \in L.$

Claim 3.0.1. *L is regular iff $\sim_L$ has a finite number of equivalence classes.*

Proof. Proof of $\Longrightarrow$: Consider a regular language L. Then $\exists$ a DFA M that recognizes L.
Let $M(x)$ denote the state of M after reading in the string x.
Observe that if $M(x_1) = M(x_2)$ then $x_1 \sim_L x_2$.
Thus the number of equivalence classes is $\leq$ the number of states in M.
(Note: It can be less if M has redundant states.) □

Proof. Proof of $\Longleftarrow$: Let $C(x)$ denote the class containing x. We construct a DFA M that recognizes L.
M has one state corresponding to each class. $C(\epsilon)$ is the start state.
The state for $C(x)$ (where x is a representative from that class) accepts iff $x \in L$.
The transitions are as follows: $C(x) \xrightarrow{\sigma} C(x\sigma)$ for $\sigma \in \Sigma$.
From the definition of Nerode-equivalence, you can verify that M recognizes L. □

We can use this Claim to show that $a^n b^n$ is not regular:
Each string in the infinite set $\{a^i \,|\, i \geq 0\}$ is in a different Nerode class.

4 The Pumping Lemma

Claim: Joe Kilian thinks the Pumping Lemma is overrated.
Proof: Left as an exercise to the reader.

"Those who cannot remember the past are condemned to repeat it." – George Santayana

The truth about DFAs: "Even if you remember the past, you are condemned to repeat."
What this means: Since a DFA M has a finite number of states, say n states, if you send a string of length $> n$ through M, it must pass through some state twice. (This is really just an application of the Pigeonhole Principle.)

The idea: Suppose there exists a string $x \in L$ (where L is the language accepted by M), such that $|x| > n$. Think of what happens as you run M on input x. At some point, you will hit a state S that you passed through earlier in the string. Then the path from the previous time to now forms a cycle in M.

Now let's break down x into three parts: w_1 is the part of the string up until you get to state S for the first time, w_2 is from there until you hit S for the second time, and w_3 is the remainder of the string. Then since w_2 is just going around the loop, you can loop as many times as you want before continuing with w_3, which will then bring you to the final accept state (since we know $x \in L$).

Thus we have found an infinite number of strings in the language L, namely anything of the form: $w_1 w_2^k w_3$, where $k \geq 0$.

In the interest of time and clarity, please consult
`http://en.wikipedia.org/wiki/Pumping_lemma_for_regular_languages`
for a formal definition and good explanation of the Pumping Lemma.

5 A Powerful PFA

aka **"The section that tells you how to do problem #5 on the homework"**

We can make a PFA that recognizes $a^n b^n$!
Here's the setup:

- Can we ensure that a string is of the form $a^n b^m$? Yes.
- Can we ensure that $n \equiv m \pmod{100}$? Yes.
- So the only possibilities left are either $n = m$ or $|n - m| \geq 100$.
- Idea: We can detect such a large difference fairly accurately using probability.

Let's play a game: Team A,B vs Team C,D. This is a game of Elimination.
Each player has a "personality" :

- Player A does well when there are not too many "a"s, i.e. $n \leq m - 100$.
- Player B does well when there are not too many "b"s, i.e. $m \leq n - 100$.
- Players C and D are apathetic, and don't care how many of each symbol there are.

So here's the game:

- Read through an input string $a^n b^m$, one character at a time.
- Each time you read "a": eliminate A with prob $\frac{3}{4}$, and eliminate C,D each with prob $\frac{1}{2}$.
- Each time you read "b": eliminate B with prob $\frac{3}{4}$, and eliminate C,D each with prob $\frac{1}{2}$.

When you finish reading the string, we say a team has survived if at least one of its members has survived. If exactly one team survives, that team wins.

Now let's evaluate each player's chance of survival:

- $Pr(A \text{ survives}) = (1/4)^n = \frac{1}{2^{2n}}$
- $Pr(B \text{ survives}) = (1/4)^m = \frac{1}{2^{2m}}$
- $Pr(C \text{ survives}) = (1/2)^{m+n} = \frac{1}{2^{m+n}}$
- $Pr(D \text{ survives}) = (1/2)^{m+n} = \frac{1}{2^{m+n}}$
- $Pr(AB \text{ survives}) = \frac{1}{2^{2n}} + \frac{1}{2^{2m}} - (1/4)^{m+n}$
- $Pr(CD \text{ survives}) = \frac{1}{2^{m+n}} + \frac{1}{2^{m+n}} - (1/4)^{m+n}$

Assuming we do the preliminary testing described in "Setup" above, there are three cases:

- Case $n = m$: $Pr(AB \text{ survives}) = Pr(CD \text{ survives})$.
- Case $n \leq m - 100$: Player A is much, much more likely to survive than any of the others, so $Pr(AB \text{ survives}) \gg Pr(CD \text{ survives})$.
- Case $m \leq n - 100$: Player B is much, much more likely to survive than any of the others, so $Pr(AB \text{ survives}) \gg Pr(CD \text{ survives})$.

Intuitively, if $n = m$ then each team has the same chance of survival, but if $n \neq m$ then Team AB has a much, much better chance of survival.

Okay, so let's take another look at this game. We play the game with an input string of the form $a^n b^m$, where $n \equiv m \pmod{100}$. Notice that a vast majority of the time, all the players will be eliminated pretty quickly. But suppose that a "miracle" happens, and one of the teams actually survives. Well, if $n = m$ then we can be pretty certain that Team AB is the one that survived. But if $n \neq m$, it's really a 50/50 guess as to which team survived.

So here's the overall idea: Play this game a lot of times on the same input string, enough times that you get several "miracles." If Team AB wins in all of those "miracle" games, then you can be almost sure that the string is not of the form $a^n b^n$. However, if Team AB wins some and Team CD wins some, then you can be almost sure that the string is of the proper form.

This leads to the creation of a PFA that can decide membership in the language $a^n b^n$:

- Given an input string x, test if it is of the form $a^n b^m$. If not, reject.
- Then check if $n \equiv m \pmod{100}$. If not, reject.
- Now simulate the game 100 times. If there's more than one "miracle" and they all end with Team AB winning, reject. If there's more than one "miracle" and Team CD wins sometimes, accept. Otherwise, randomly accept or reject with probability 1/2 each.

For any input string x, this PFA accurately decides membership with probability $> \frac{1}{2}$.

6 Potpourri and Miscellany

6.1 Other Questions

- Is the language L represented by the DFA M non-trivial? (i.e. $L \notin \{\emptyset, \Sigma^*\}$)
  Yes, we can check that by seeing if an (non-)accept state can be reached from the start state.
- Given DFAs M_1, M_2 for languages L_1, L_2, can we check whether $L_1 = L_2$?
  Yes, just check if $L_1 \cap \overline{L_2} = \emptyset$ and $\overline{L_1} \cap L_2 = \emptyset$.
- How efficient are our reductions?

6.2 Example: DFA harder than RegExp

Let $L = \{x \mid \text{some character in } x \text{ appears twice}\}$.

- Building an NFA for L can be done easily in $O(n)$ states, where $n = |\Sigma|$.
  (See Figure 1, top of next page.)
- Making a RegExp for L is also easy in $O(n)$: $\bigcup_{\sigma \in \Sigma} \Sigma^* \sigma \Sigma^*$.
- However, the size of a DFA to recognize L must be exponential. Why?
  There are 2^n distinct Nerode Equivalence Classes - one for each subset of Σ.

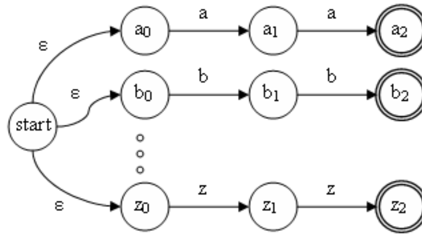

Figure 1: an NFA with O(n) states

6.3 Example: Requiring a big NFA

Let $L = \{x \mid \text{each character } \sigma \in \Sigma \text{ appears exactly once in } x\}$.
How big is the NFA M for this language? Answer: It must be exponential.

- For each string $w = w_1 w_2 \in L$ (w_1 is the first half of w, w_2 the second half), there must be a halfway-state S_w in M, i.e. a state that is reachable from reading in w_1, and from there you can reach an accept state from reading w_2.
- Exercise: Each possible subset represented by w_1 (the subsets of size $n/2$) requires a different halfway-state.
- Thus M has at least $\binom{n}{n/2} = \Theta(n^{n/2})$ states.

6.4 Extending the Model

Now we will look at a more general model of a DFA, described by (1) the observables, and (2) the possible actions or behavior of the machine.

6.4.1 2-Way Machines

- Observables: state, finite range of input symbols (e.g. can view 3 consecutive characters)
- Behavior: change state, can go in either direction in input string

Note: A 2-way DFA can also be referred to as a read-only Turing machine. (If you don't know what that means, don't worry – we'll get to that soon.)

Surprise: This is actually equivalent to a normal DFA, i.e. it can still only recognize regular languages! However, if you allow the machine to have two different pointers in the input string, it then becomes more powerful.

Exercise: Show that such a machine with two pointers can recognize $a^n b^n$.

6.4.2 Counting Machines

- Observables: state, input symbol, whether counter is at 0
- Behavior: change state, move forward in string, increment or decrement counter (obeys some predetermined rule if instructed to decrement counter when it is at 0)

Note: We could not allow such a machine to observe the exact value of the counter, since the amount of space required would be unbounded.