Harvard CS 121 and CSCI E-121

Lecture 7: The Pumping Lemma and Non-Regular Languages

Harry Lewis

September 24, 2013

- **Reading:** Sipser, §1.4.

Finding a NonRegular Language

Strategy:

(1) Identify some property that all regular languages have

(2) Show that the candidate language does not have property (1)

Pumping Lemma (Basic Version)

If L is regular,
then there is a number p (the pumping length)
such that
every string $s \in L$ of length at least p
can be divided into $s = xyz$, where $y \neq \varepsilon$ and
for every $n \geq 0$, $xy^n z \in L$.

$n = 1$

x	y	z

$n = 0$

x	z

$n = 2$

x	y	y	z

$\ldots$

- Why is the part about p needed?
- Why is the part about $y \neq \varepsilon$ needed?

Pumping Lemma (Basic Version)

($\forall$) If L is regular,
($\exists$) then there is a number p (the pumping length)
such that
($\forall$) every string $s \in L$ of length at least p
($\exists$) can be divided into $s = xyz$, where $y \neq \varepsilon$ and
($\forall$) for every $n \geq 0$, $xy^n z \in L$.

$n = 1$ | x | y | z |

$n = 0$ | x | z |

$n = 2$ | x | y | y | z |

$\ldots$

- Why is the part about p needed?

- Why is the part about $y \neq \varepsilon$ needed?

Pumping Lemma Example

- Consider

$$L = \{x : x \text{ has an even \# of } a\text{'s and an odd \# of } b\text{'s}\}$$

- Since L is regular, pumping lemma holds.

  (i.e, every sufficiently long string s in L is "pumpable")

- For example, if $s = aab$, we can write $x = \varepsilon$, $y = aa$, and $z = b$.

Pumping the even a's, odd b's language

- Claim: L satisfies pumping lemma with pumping length $p = 4$.
- Proof:
- **Q:** Can the Pumping Lemma be used to prove that L is regular?

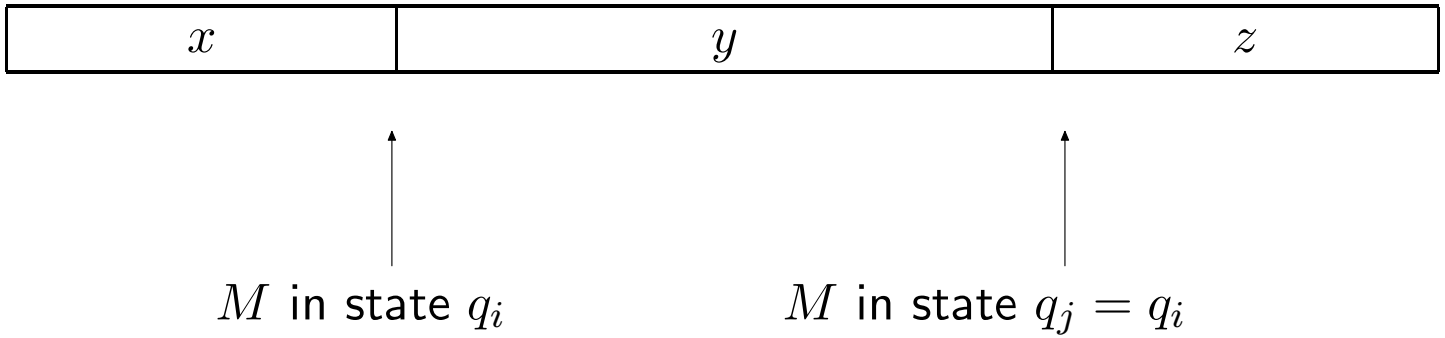
Proof of Pumping Lemma

(Another fooling argument)

- Since L is regular, there is a DFA M recognizing L.
- Let $p =$ # states in M.
- Suppose $s \in L$ has length $l \geq p$.
- M passes through a sequence of $l + 1 > p$ states while accepting s (including the first and last states): say, $q_0, \ldots, q_l$.
- Two of these states must be the same: say, $q_i = q_j$ where $i < j$

Pumping, continued

- Thus, we can break s into x, y, z where $y \neq \varepsilon$ (though x, z may equal ε):



- If more copies of y are inserted, M "can't tell the difference," i.e., the state entering y is the same as the state leaving it.

- So since $xyz \in L$, then $xy^n z \in L$ for all n.

Proof also shows (why?):

- We can take p = # states in smallest DFA recognizing L.

- Can guarantee division $s = xyz$ satisfies $|xy| \leq p$ (or $|yz| \leq p$).

Use PL to Show Languages are NOT Regular

Claim: $L = \{a^n b^n : n \geq 0\} = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$ is not regular.

Proof by contradiction:

- Suppose that L is regular.
- So L has some pumping length $p > 0$.
- Consider the string $s = a^p b^p$. Since $|s| = 2p > p$, we can write $s = xyz$ for some strings x, y, z as specified by lemma.
- Claim: No matter how s is partitioned into xyz with $y \neq \varepsilon$, we have $xy^2z \notin L$.
- This violates the conclusion of the pumping lemma, so our assumption that L is regular must have been false.

Strings of exponential lengths are a nonregular language

Claim: $L = \{w : |w| = 2^n \text{ for some } n \geq 0\}$ is not regular.

Proof:

- Suppose L satisfied the pumping lemma with pumping length p.
- Choose any string $s \in L$ of length greater than p, say $|s| = 2^n$. By pumping lemma, write $s = xyz$.
- Let $|y| = k$. Then $2^n - k, 2^n, 2^n + k, 2^n + 2 \cdot k, \ldots$ are all powers of two.
- This is impossible. (Why?) QED

"Regular Languages Can't Do Unbounded Counting"

Claim: $L = \{w : w$ has the same number of a's and b's$\}$ is not regular.

Proof #1:

- Use pumping lemma on $s = a^p b^p$ with $|xy| \leq p$ condition.

"Regular Languages Can't Do Unbounded Counting"

Claim: $L = \{w : w$ has the same number of a's and b's$\}$ is not regular.

Proof #1:

- Use pumping lemma on $s = a^p b^p$ with $|xy| \leq p$ condition.

Proof #2:

- If L were regular, then $L \cap a^* b^*$ would also be regular.

Reprise on Regular Languages

Which of the following are necessarily regular?

- A finite language
- A union of a finite number of regular languages
- A union of a countable number of regular languages
- $\{x : x \in L_1$ and $x \notin L_2\}$, L_1 and L_2 are both regular
- A cofinite language (a set is *cofinite* if its complement is finite)
- The reversal of a regular language

Algorithmic questions about regular languages

Given X = a regular expression, DFA, or NFA,
how could you tell if:

- $x \in L(X)$, where x is some string?
- $L(X) = \emptyset$?
- $x \in L(X)$ but $x \notin L(Y)$?
- $L(X) = L(Y)$, where Y is another RE/FA?
- $L(X)$ is infinite?
- There are infinitely many strings that belong to both $L(X)$ and $L(Y)$?

A Final Note on Regular Languages and Finite Automata

For any regular language L, there are infinitely many different finite automata accepting L, and infinitely many different regular expressions representing L. Why?

A Final Note on Regular Languages and Finite Automata

For any regular language L, there are infinitely many different finite automata accepting L, and infinitely many different regular expressions representing L. Why?

For any regular language L, there is a **unique** *minimal* finite automaton M such that $L(M) = M$.

That is, $L(M) = L$, and for any other finite automaton M' such that $L(M') = L$, either M' has more states than M or its state diagram is isomorphic to that of M.

The minimal equivalent finite automaton can be found constructively.

Minimizing DFAs

Finding the minimal equivalent DFA:

- Let M be a DFA
- Say that states p, q of M are *distinguishable* if there is a string w such that exactly one of $\delta^*(p, w)$ and $\delta^*(q, w)$ is final.
- Start by dividing the states of M into two equivalence classes: the final and non-final states

Minimizing DFAs, continued

- Break up the equivalence classes according to this rule: If p, q are in the same equivalence class but $\delta(p, \sigma)$ and $\delta(q, \sigma)$ are not equivalent for some $\sigma \in \Sigma$, then p and q must be separated into different equivalence classes
- When all the states that must be separated have been found, form a new and finer equivalence relation
- Repeat
- How do we know that this process stops?

Generalizations of FA

Can add:

- probabilistic transitions (like Markov chains)
- outputs at each state
- rewards at each state
- infinite state spaces

Often referred to as “state machines”.