

Lecture 5: Pumping Lemma and Closure Properties

David Dill

Department of Computer Science

Outline

- Pumping Lemma
- Closure properties.

Pumping Lemma

Primary application: Proving that certain languages are *not* regular.

Lemma: For every regular language L , there exists a positive constant n such that for every string w in L such that $|w| \geq n$, there exist strings x, y, z such that $w = xyz$ with the following properties:

- $y \neq \epsilon$
- $|xy| \leq n$
- $xy^kz \in L$ for all $k \geq 0$.

Understanding the Pumping Lemma

The pumping lemma is a little difficult to understand because of all the “alternating quantifiers” ($\forall L \exists n \forall w \exists x, y, z \forall k \dots$)

With such theorems, it is often helpful to think of a game with two players. Player \exists is trying to make the theorem true, player \forall is trying to falsify it.

For applications of the Pumping Lemma, it’s most useful to treat L as the “game board,” which is given, and take the game from there.

Player \exists : Chooses the constant n .

player \forall : Chooses $w \in L$ such that $|w| \geq n$.

Player \exists : Chooses $xyz = w$ such that $y \neq \epsilon$ and $|xy| \leq n$.

player \forall : Chooses k .

player \forall wins if $xy^kz \notin L$.

The pumping lemma says: If L is regular, Player \exists has a winning strategy.

(Important note: If L is not regular, Player \exists may *still* have a winning strategy.

This version of the Pumping Lemma is only necessary for regularity, not sufficient.)

Example Application of Pumping Lemma

Theorem: $L =$ “All strings over $\Sigma = \{0, 1\}$ with the same number of 0’s and 1’s” is not regular.

Proof: We prove by contradiction, using the PL. Suppose L were regular.

Let n be the “pumping constant” of the PL. [This is Player \exists ’s move. player \forall has no idea what n is, but gives a general rule for the next move for any n .]

Consider the string $w = 0^n 1^n \in L$. [This is player \forall ’s move, which works for any n .]

By the PL, there must be $xyz = 0^n 1^n$, such that $|xy| \leq n$ and $y \neq \epsilon$, and $xy^k z \in L$ for all $k \geq 0$. [Player \exists ’s move.]

So xy will consist of all 0s and y will be a string of at least one 0. [We don’t know what xyz are, but we know these properties hold if Player \exists made a legal move.]

If $k \neq 1$, then $xy^k z$ will have a different number of 0s than xyz and the same number of 1s, so $xy^k z \notin L$, contradicting the previous statement above. [player \forall can win by choosing $k = 0$. Then that will delete at least one 0, because $y \neq \epsilon$.]

Hence, the assumption that L is regular must be false.

Another non-regular language

$\{0^i 1^i \mid i \geq 0\}$ is not regular. This has basically the same proof as the previous theorem, and is quite useful to know.

Proof of the Pumping Lemma

Let L be any regular language. We know there is a DFA D that accepts the same language. Let n be the number of states in that DFA.

Let w be any string in L such that $|w| \geq n$

Let $u = a_1 a_2 \dots a_n$ be the prefix of w of length n .

Consider the sequence of states $p_0 p_1 \dots p_n$ such that $p_0 = q_0$ and $\hat{\delta}(p_0, a_1 a_2 \dots a_i) = p_i$.

There are $n + 1$ positions in the sequence of states, so, by the pigeonhole principle, there is at least one state that appears at two distinct positions; call the first position i and the second j .

So, $0 \leq i < j \leq n$ and $p_i = p_j$.

Pumping lemma proof, cont.

Let $x = a_1 a_2 \dots a_i$, let $y = a_{i+1} \dots a_j$ and $z = a_{j+1} \dots a_{|w|}$ (in fact, y may be only one symbol and z can be empty, but you get the idea).

So $w = xyz$ and $\hat{\delta}(q_0, x) = p_i$, $\hat{\delta}(p_i, y) = p_j = p_i$ and $\hat{\delta}(p_j, z) \in F$.

$|xy| \leq n$ (since $j \leq n$). $y \neq \epsilon$ because $j > i$.

Then every string of the form $xy^k z$ for $k \geq 0$ is in L , because $\hat{\delta}(q_0, xy^k) = p_i$, hence $\hat{\delta}(q_0, xy^k z) = \hat{\delta}(p_i, z) \in F$.

Closure Properties

A "closure property" is a theorem that shows that certain operations preserve some property. People often say that a set is closed under certain operations.

For example, the integers are closed under addition, subtraction, and multiplication, but not division. So a closure property is that addition of integers gives an integer.

Closure properties are generally of interest to mathematics. They provide powerful ways to prove that certain sets have a property, and, sometimes more importantly, that the sets do *not* have the property.

That's why we're interested in closure properties of regular languages.

We can use all the different representations we have (reg exprs, DFAs, NFAs e-NFAs) to prove closure properties.

Regular Operators

We can use all the different representations we have (reg exprs, DFAs, NFEs e-NFAs) to prove closure properties.

Theorem Regular sets are closed under the operations: union, concatenation, Kleene closure.

Proof: Suppose L_1 and L_2 are regular languages.

Then there exist regular expressions R_1 and R_2 such that $L(R_1) = L_1$ and $L(R_2) = L_2$.

But then $L(R_1 + R_2) = L_1 \cup L_2$, $L(R_1 R_2) = L_1 L_2$, and $L(R_1^*) = L_1^*$. So each of these languages is regular as well.

Boolean Operations

Def. The *complement* of a language (relative to an alphabet Σ):

$$\overline{L} = \Sigma^* - L = \{x \in \Sigma^* \mid x \notin L\}.$$

The *Boolean operations* are union, intersection, and complement.

Theorem: The regular languages are closed under Boolean operations.

Proof:

We've already got union.

Complement:

Build the DFA D . Define D_C as the same DFA, but complement the set of final states ($F_C = Q - F$).

It's easy to see that this DFA accepts the complement of the language.

Note: This really requires the δ to be a total function (no missing arrows), which is how we defined it.

Note: This absolutely does not work for NFAs!!!!

Intersection

Mathematician's method: If we know how to do union and complement, we can use de Morgan's law to do intersection, since:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

Product construction: There is a different, more direct approach to intersection that is so important that we should talk about it. It is the *product construction*.

Intuitively, the product combines two automata by working on all the pairs of states, one from each automaton.

(Remember the *cartesian product* of two sets

$$S \times T = \{(x, y) \mid x \in S \wedge y \in T\}.$$

Given DFAs $D_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ $D_2 = (Q_2, \Sigma, \delta_2, s_0, F_2)$

The product is $(Q_1 \times Q_2, \Sigma, \delta, (q_0, s_0), F_1 \times F_2)$

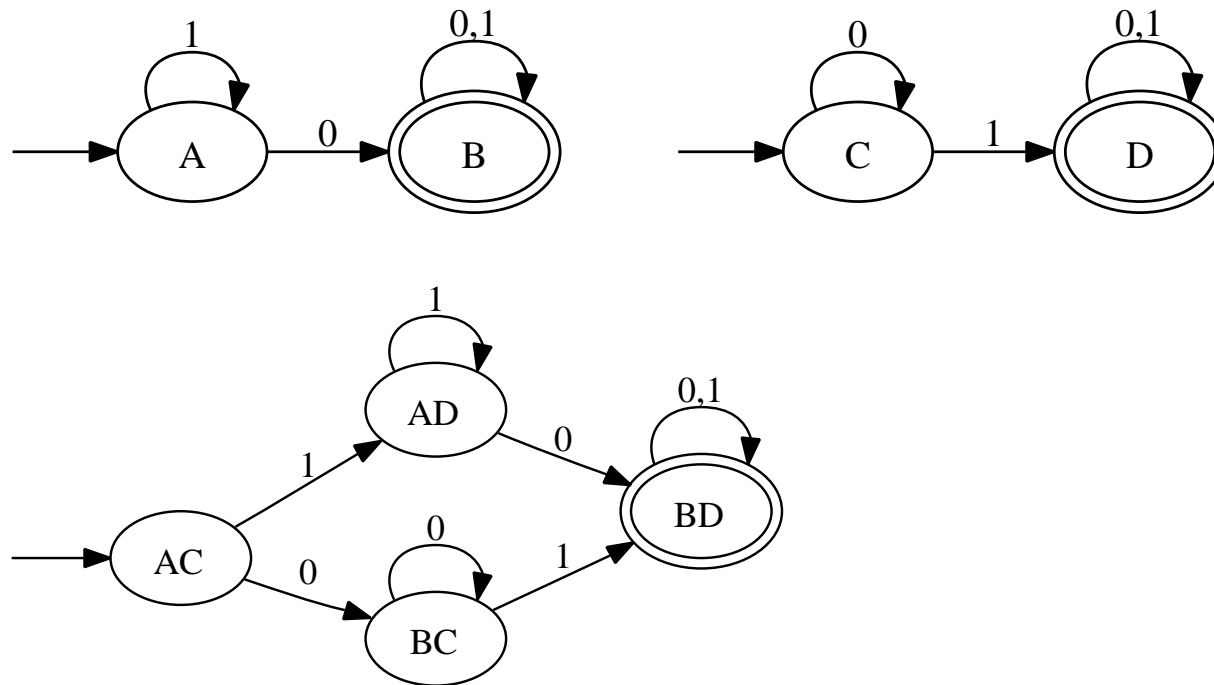
where $\delta((q, r), a) = (\delta_1(q, a), \delta_2(r, a))$ for all $q \in Q_1$ and $r \in Q_2$.

This DFA accepts $L(D_1) \cap L(D_2)$.

Interesting point: Essentially the same construction works for NFAs.

Product Construction Example

Consider intersection of “at least one 0” and “at least one 1”.



Language Difference

$$L_1 - L_2 = L_1 \cap \overline{L_2}.$$

If D_2 is a DFA for L_2 , you can do this efficiently. Or you can do a product construction where $F = F_1 \times (Q_2 - F_2)$.

Reversal

Reverse of a string: if $x = a_1a_2 \dots a_n$, then

$$x^R = a_n \dots a_2a_1.$$

$$L^R = \{x^R \mid x \in L\}$$

Theorem If L is regular, so is L^R .

Proof: Reversing the regular expression for L gives a regular expression for L^R .

Formal approach: Define R^R recursively on regular expression, prove that L^R is $L(R^R)$ by induction on the structure of regular expressions.

String Homomorphisms

Start with function $h: \Sigma_1 \rightarrow \Sigma_2^*$. Then extend it to strings:

Base: $h(\epsilon) = \epsilon$

Induction: $h(xa) = h(x)h(a)$.

So, if $h(0) = ab$ and $h(1) = \epsilon$ then $h(0110) = abab$

h can also be applied to entire languages: $h(L) = \{h(x) \mid x \in L\}$.

Theorem: If $L \subseteq \Sigma_1^*$ is regular and $h: \Sigma_1 \rightarrow \Sigma_2^*$ is a string homomorphism, then $h(L)$ is regular.

Construction: There is a regular expression R for L . Substitute $h(a)$ for each occurrence of each a in R .

Formally: Recursive definition of $h(R)$, proof by induction on structure of regular expressions that $L[h(R)] = h(L)$.

Inverse Homomorphism

$$h^{-1}(L) = \{x \mid h(x) \in L\}.$$

Consider h that deletes “ c ”: $h(a) = a$, $h(b) = b$, $h(c) = \epsilon$.

Consider $L = (abc)^*$. What is $h(L)$?

What is $h^{-1}[(ab)^*]$? Answer: $c^*(c^*ac^*bc^*)^*$ – *not the same language*.

$h^{-1}(L)$ is the *largest* language L' such that $h(L') = L$.

Closure under Inverse Homomorphism

Theorem: If $L \subseteq \Sigma_2^*$ is regular and h is a homomorphism from $\Sigma_1 \rightarrow \Sigma_2^*$, then $h^{-1}(L)$ is regular.

Construction:

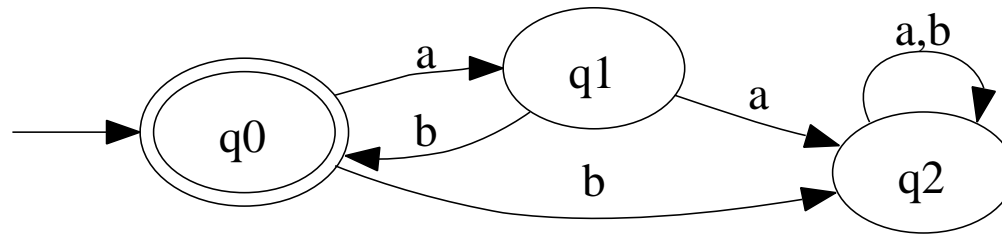
Let $(Q, \Sigma_2, \delta, q_0, F)$ be a DFA accepting L . We construct $(Q, \Sigma_1, \gamma, q_0, F)$ such that

$$\gamma(q_i, a) = \hat{\delta}(q_i, h(a))$$

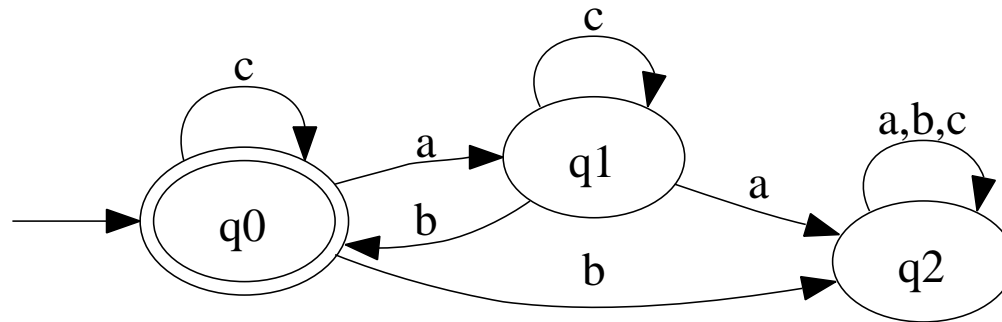
Rest of proof: Show $\hat{\gamma}(q_i, x) = \hat{\delta}(q_i, h(x))$ for all $x \in \Sigma_1^*$.

Example application of Inverse Homomorphism Construction

Try it on “delete c ” with $(ab)^*$. Here is a DFA for $(ab)^*$



Here is the result of the construction for the inverse homomorphism:



This is the same language as before.

Closure Properties for Non-Regularity

Proof by contradiction: If we can construct a non-regular language using regularity-preserving operations, one of the original languages must have been non-regular.

Example: The language L over $\Sigma = \{0, 1, 2\}$ having equal numbers of 0's and 1's is not regular.

proof: Suppose the language were regular. Define h such that $h(0) = 0$, $h(1) = 1$, and $h(2) = \epsilon$. Then $h(L)$ would be regular. But $h(L)$ is the language over $\{0, 1\}$ having equal numbers of 0's and 1's, proved non-regular above, so L must not be regular.

Example: The language of strings over $\Sigma = \{0, 1\}$ having different numbers of 0's and 1's is not regular.

proof By contradiction: Suppose the language were regular. Then its complement would be, too, since regular languages are closed under complementation.

But the complement of this language is the language of strings having the same number of 0's and 1s, which was proved non-regular earlier in the lecture using the pumping lemma.

So the language in question must not be regular.