6.045: Automata, Computability, and Complexity
Or, Great Ideas in Theoretical Computer Science
Spring, 2010

Class 5

Nancy Lynch

Today

- Non-regular languages
- Today's topics:
  - Existence of non-regular languages
  - Showing some specific languages aren't regular
  - The Pumping Lemma
  - Examples
  - Algorithms that answer questions about FAs.
- Reading: Sipser, Sections 1.4, 4.1.
- Next:
  - Computability theory
  - Readings:
    - Sipser Chapter 3
    - The Nature of Computation, Chapter 8
    - GITCS notes, lecture 4

Existence of Non-Regular Languages

Existence of non-regular languages

- Theorem: There is a language over $\Sigma = \{ 0, 1 \}$ that is not regular.
- (Works for other alphabets too.)
- Proof:
  - Recall, a language is any (finite or infinite) set of (finite) strings.
  - It turns out that there are many more sets of finite strings than there are DFAs; so just based on cardinality, there must be some non-regular languages.
  - But, there are infinitely many sets of strings, and infinitely many DFAs---so what does it mean to say that one of these is “more” than the other?
  - Answer: There are different kinds of infinities:
    - Countably infinite sets, like the natural numbers or the integers.
    - Uncountably infinite sets, like the reals.
    - Also, different sizes of uncountable infinities.

Existence of non-regular languages

- Theorem: There is a language over $\Sigma = \{ 0, 1 \}$ that is not regular.
- Proof:
  - Follows from two claims:
  - Claim 1: The set of all languages over $\Sigma = \{ 0, 1 \}$ is uncountable, that is, it cannot be put into one-to-one correspondence with N (natural numbers).
  - Claim 2: The set of regular languages is countable.

Claim 1

- Claim 1: The set of all languages over $\Sigma = \{ 0, 1 \}$ is uncountable, that is, it cannot be put into one-to-one correspondence with N.
- Proof of Claim 1: By contradiction.
  - Suppose it is countable.
  - Then we can put the set of all languages in one-to-one correspondence with N, e.g.:

0 ……. $\varnothing$ L_0

1 ……. { 0 } L_1

2 ……. All even-length strings (an infinite language) L_2

3 ……. All strings containing at least one 0 L_3

Etc.

All (finite and infinite) sets of (finite) strings must appear in this list.

Claim 1

- Claim 1: The set of all languages over $\Sigma = \{ 0, 1 \}$ is uncountable, that is, it cannot be put into one-to-one correspondence with N (the natural numbers).
- Proof, cont'd:
  - Clarify:
    - Σ^* is the set of all (finite) strings over $\Sigma = \{ 0, 1 \}$.
    - $P(\Sigma^*)$ is the set of all sets of strings, or languages, over Σ.
    - Right column lists all languages, that is, all elements of $P(\Sigma^*)$.
  - Σ^*, the set of all finite strings, is countable:
    - We can list all finite strings in order of length, put them in one-to-one correspondence with N.
    - E.g., ε, 0, 1, 00, 01, 10, 11, 000,…
  - Since there is a correspondence between N and Σ^*, and we assumed one between N and $P(\Sigma^*)$, there must be a correspondence between Σ^* and $P(\Sigma^*)$, e.g.:

Claim 1

ε ……. $\varnothing$ L_0

0 ……. { 0 } L_1

1 ……. All even-length strings (an infinite language) L_2

00 ……. All strings containing at least one 0. L_3

Etc.

- Call the correspondence f, so we have $f(\varepsilon) = L_0$, $f(0) = L_1$, $f(1) = L_2$, etc.

- Now define D, the diagonal set of strings:

  $D = \{ w \in \Sigma^* \mid w \text{ is not in } f(w) \}$

- Examples:
  - ε is in D, because ε is not in $\varnothing$
  - 0 is not in D, because 0 is in { 0 }
  - 1 is in D, because 1 is not an even-length string.
  - 00 is not in D, because 00 contains at least one 0.

    Etc.

Claim 1

- Now the twist…
- Since the right column includes all subsets of Σ^*, D itself appears somewhere.
- That is, D = f(x) for some string x.

  x ……. D = { w | w is not in f(w) }
- Tricky question: Is this string x in D or not?
- Two possibilities:
  - If x is in D, then x is not in f(x) by definition of D, so x is not in D since D = f(x).
  - If x is not in D, then x is in f(x) by definition of D, so x is in D since D = f(x).
- Either way, a contradiction.
- Implies that no such mapping f exists.
- So there is no correspondence between N and $P(\Sigma^*)$.
- So $P(\Sigma^*)$, the set of languages over Σ, is uncountable.

Claim 2

- Claim 2: The set of regular languages is countable.
- Proof:
  - Each regular language is recognized by some DFA.
  - Each DFA has a finite description: states, start states, transitions,…
  - Can write each of these using standard names for states, without changing the language.
  - Can enumerate these "standard form" DFAs in order of length.
  - Leads to an enumeration of the regular languages.

- Since $P(\Sigma^*)$, the set of all languages, is uncountable, whereas the set of regular languages is countable, some language must be non-regular.
- In fact, by considering different kinds of infinity, one can prove that "most" languages are non-regular.

Showing specific languages are non-regular

Showing languages are non-regular

- Basic tool: Pigeonhole Principle: If you put > n pigeons into n holes, then some hole has > 1 pigeon.

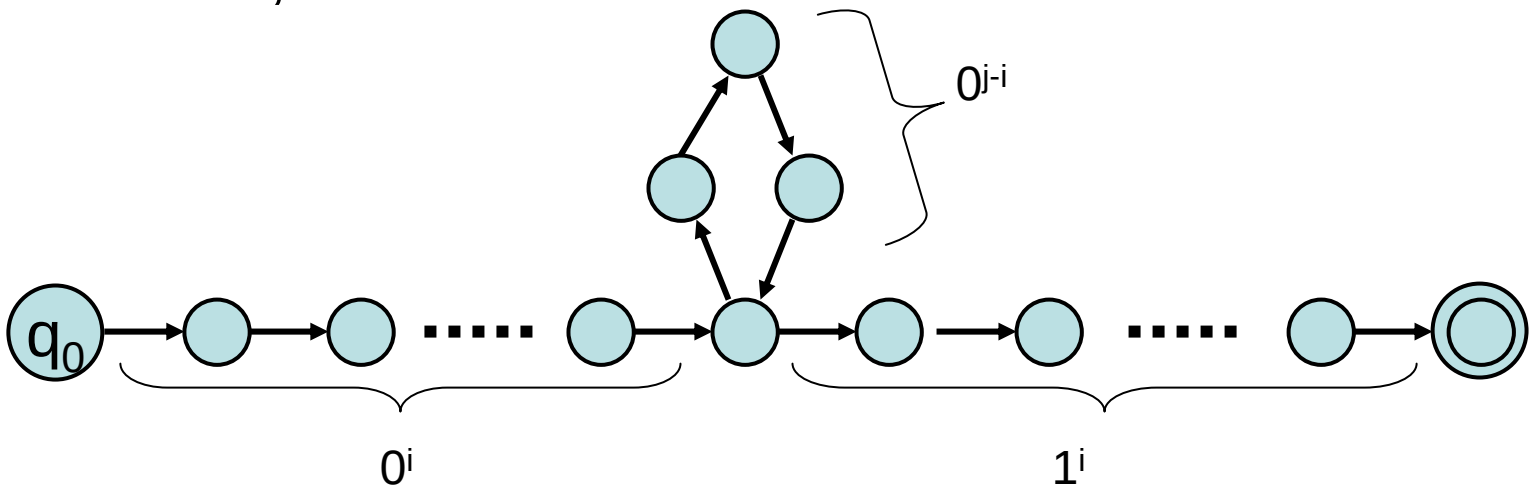
- Example 1: $L_1 = \{ 0^n1^n \mid n > 0 \}$ is non-regular
  - E.g., 0011 is in L_1, 011 is not.
  - Show by contradiction, using Pigeonhole Principle.
  - Assume L_1 is regular.
  - Then there is a DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizing L_1.
  - Now define:
    - Pigeons = all strings in 0*.
    - Holes = states in Q.
  - Put pigeon 0^i into hole $\delta^*(q_0, 0^i)$, that is, the hole corresponding to the state reached by input 0^i.

Showing languages are non-regular

- Example 1: $L_1 = \{ 0^n1^n \mid n \geq 0 \}$ is non-regular
  - Assume L_1 is regular.
  - Then there is a DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizing L_1.
  - Define:
    - Pigeons = all strings in 0*.
    - Holes = states in Q.
  - Put pigeon 0^i into hole $\delta^*(q_0, 0^i)$, that is, the hole corresponding to the state reached by input 0^i.
  - There are |Q| holes, but > |Q| pigeons (actually, infinitely many).
  - So by Pigeonhole Principle, 2 pigeons must be put in the same hole, say 0^i and 0^j with $i < j$.
  - That is, 0^i and 0^j lead to the same state.
  - Then since M accepts 0^i1^i, it also accepts 0^j1^i, which is incorrect, contradiction.

Showing languages are non-regular

- Example 1: $L_1 = \{ 0^n1^n \mid n \geq 0 \}$ is non-regular
  - Assume L_1 is regular.
  - Then there is a DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizing L_1.
  - 0^i and 0^j lead to the same state.
  - Then since M accepts 0^i1^i, it also accepts 0^j1^i, which is incorrect, contradiction.



0^i1^i leads to the final state, so 0^j1^i does also.

Showing languages are non-regular

- Example 2: $L_2 = \{ 010010001\ldots0^i1 \mid i \text{ is any positive integer } \}$ is non-regular
  - Show by contradiction, using Pigeonhole Principle.
  - Assume L_2 is regular, so there is a DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizing L_2.
  - Define:
    - Pigeons = all strings in L_2.
    - Holes = states.
  - Put pigeon string into hole corresponding to the state it leads to.
  - By the Pigeonhole Principle, two pigeons share a hole, say $01\ldots0^i1$ and $01\ldots0^j1$, where $j > i$.
  - So $01\ldots0^i1$ and $01\ldots0^j1$ lead to the same state.
  - M accepts $01\ldots0^i10^{i+1}1$.
  - So M accepts $01\ldots0^j10^{i+1}1$, incorrect, contradiction.

Showing languages are non-regular

- Example 3: $L_3 = \{ w\,w \mid w \in \{ 0, 1 \}^* \}$ is non-regular
  - Show by contradiction, using Pigeonhole Principle.
  - Assume L_3 is regular, so there is a DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizing L_3.
  - Define:
    - Pigeons = strings of the form 0^i1 where i is a nonnegative integer; that is, 1, 01, 001,…
    - Holes = states.
  - Put pigeon string into hole corresponding to the state it leads to.
  - By the Pigeonhole Principle, two pigeons share a hole, say 0^i1 and 0^j1, where $j > i$.
  - So 0^i1 and 0^j1 lead to the same state.
  - M accepts 0^i10^i1.
  - So M accepts 0^j10^i1, incorrect, contradiction.

The Pumping Lemma

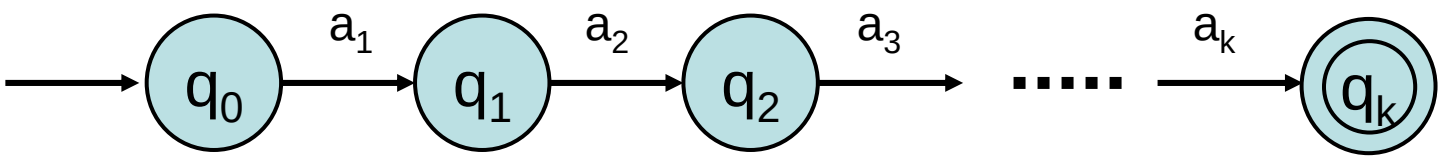
Pumping Lemma

- Use Pigeonhole Principle (PHP) to prove a general result that can be used to show many languages are non-regular.
- Theorem (Pumping Lemma):
  - Let L be a regular language, recognized by a DFA with p states.
  - Let $x \in L$ with $|x| \geq p$.
  - Then x can be written as $x = u\ v\ w$ where $|v| \geq 1$, so that for all $m \geq 0$, $u\ v^m\ w \in L$.
  - In fact, it is possible to subdivide x in a particular way, with the total length of u and v being at most p: $|\ u\ v\ | \leq p$.

- That is, we can take any sufficiently long word in the language, and find some piece that can be added in any number of times to get other words in the language ("pumping up").
- Or, we could remove the piece ("pumping down").
- And this piece could be chosen to be near the beginning of the word.

Pumping Lemma

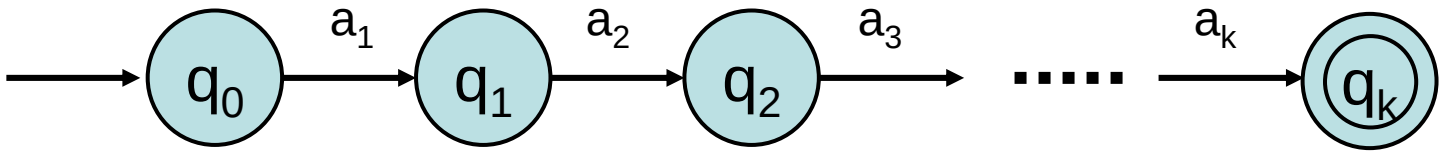
- Theorem (Pumping Lemma):
  - Let L be a regular language, recognized by a DFA with p states.
  - Let $x \in L$ with $|x| \geq p$.
  - Then x can be written as $x = u\ v\ w$ where $|v| \geq 1$, so that for all $m \geq 0$, $u\ v^m\ w \in L$.
- Proof (of the basic lemma):
  - Consider $x \in L$ with $|x| \geq p$.
  - Write $x = a_1\ a_2\ a_3 \ldots a_k$ in L, where $k \geq p$.
  - Suppose x passes through states $q_0, q_1, \ldots, q_k$, where q_0 is the start state and q_k is an accept state.



  - Since there are at least p+1 state occurrences and M has only p states, two state occurrences must be the same, by PHP.
  - Say $q_i = q_j$ for some $i < j$.

Pumping Lemma

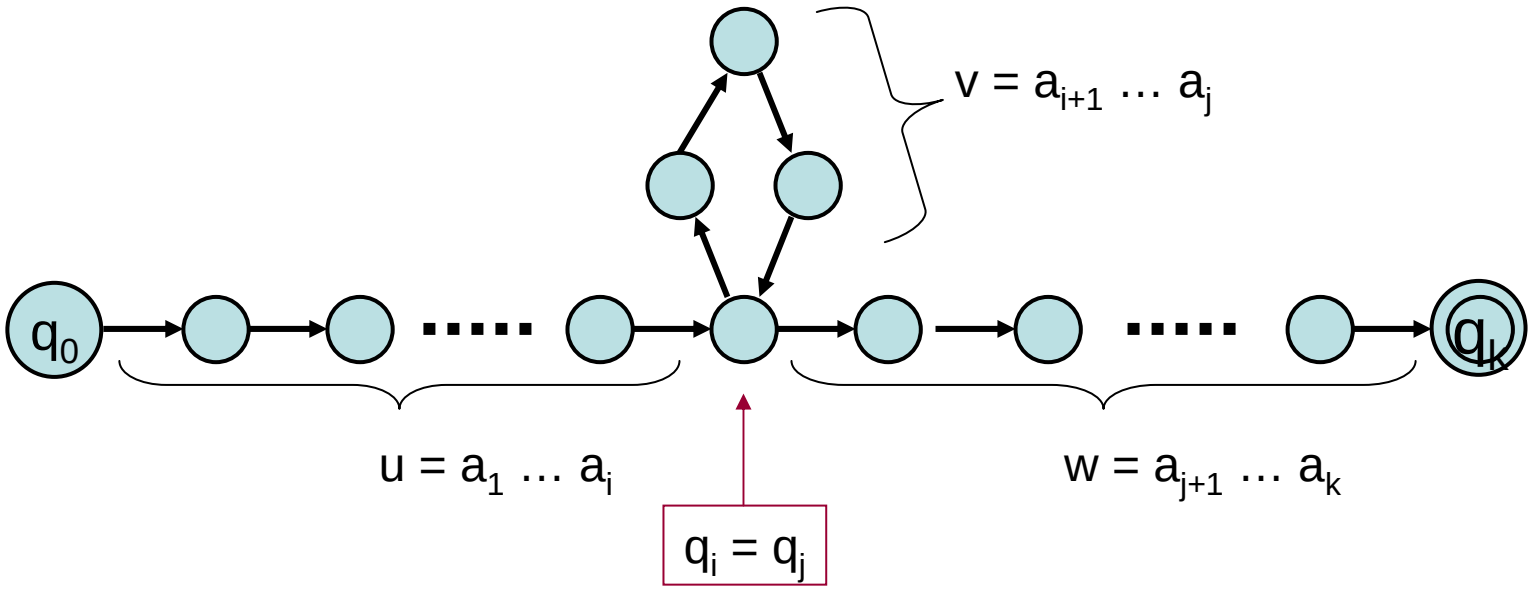
- Theorem (Pumping Lemma):
  - Let L be a regular language, recognized by a DFA with p states.
  - Let $x \in L$ with $|x| \geq p$.
  - Then x can be written as $x = u\ v\ w$ where $|v| \geq 1$, so that for all $m \geq 0$, $u\ v^m\ w \in L$.
- Proof:
  - Assume $x = a_1\ a_2\ a_3 \ldots a_k$ in L, where $k \geq p$.



  - $q_i = q_j$, $i < j$.
  - Write $u = a_1 \ldots a_i$, $v = a_{i+1} \ldots a_j$, and $w = a_{j+1} \ldots a_k$.
  - Claim this works:
    - $x = u\ v\ w$, obviously.
    - $|v| = |\ a_{i+1} \ldots a_j\ | \geq 1$, since $i < j$.
    - $u\ v^m\ w$ is accepted, since it follows the loop m times (possibly 0 times).

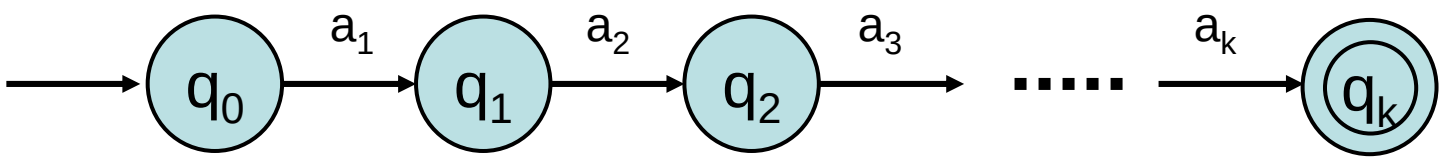
The loop

- u v^m w is accepted, since it follows the loop m times (possibly 0 times).



Getting the extra condition

- Theorem (Pumping Lemma):
  - Let L be a regular language, recognized by a DFA with p states.
  - Let $x \in L$ with $|x| \geq p$.
  - Then x can be written as $x = u\ v\ w$ where $|v| \geq 1$, so that for all $m \geq 0$, $u\ v^m\ w \in L$.
  - In fact, it is possible to subdivide x in a particular way, with the total length of u and v being at most p: $|\ u\ v\ | \leq p$.
- Proof:
  - Consider $x \in L$ with $|x| \geq p$.
  - Write $x = a_1\ a_2\ a_3 \ldots a_k$ in L, where $k \geq p$.
  - Suppose x passes through states $q_0, q_1, \ldots, q_k$.



  - Two state occurrences must be the same, by PHP.
  - We can choose these two occurrences to be among the first p+1.
  - Then $|\ u\ v\ | \leq p$.

Example 1, revisited

- $L_1 = \{ 0^n 1^n \mid n \geq 0 \}$ is non-regular.
- Suppose there is a DFA for L_1 with p states.
- We pick a particular word x in L_1 and pump it to get a contradiction.
- Choose $x = 0^p 1^p$, where p is the number of states.
- Then the Pumping Lemma says that x can be written as u v w, with $|v| \geq 1$, so that u v v w is also in L_1.
  - We're using m = 2 here.
- We get a contradiction, by considering three cases:
  - v consists of 0s only: Then u v v w contains at least one extra 0, the same 1s, can't match.
  - v consists of 1s only: At least one extra 1, can't match.
  - v consists of a mix of 0s and 1s: Then u v v w contains a 1 before a 0, so u v v w can't be in L_1.

Example 3, revisited

- $L_3 = \{ w\,w \mid w \in \{ 0, 1 \}^* \}$ is non-regular.
- Suppose there is a DFA for L_3 with p states.
- Pick a word x in L_3 and pump it to get a contradiction.
- Choose $x = 0^p\,1\,0^p\,1$, where p is the number of states.
- Pumping Lemma says that x can be written as u v w, with $|v| \geq 1$, so that $u\,v^m\,w$ is also in L_3, for every m.
- But so what?
  - The PL might give us v = x, u = w = ε.
  - Then adding in v any number of times, or removing v, yields a string in L_3.
  - E.g., if x = 001001, and v = x, then u v v w = 001001001001, which is in L_3 .
  - No contradiction here.

Example 3, revisited

- $L_3 = \{ w\,w \mid w \in \{ 0, 1 \}^* \}$ is non-regular.
- Choose $x = 0^p\,1\,0^p\,1$, where p is the number of states.
- Pumping Lemma says that x can be written as u v w, with $|v| \geq 1$, so that $u\,v^m\,w$ is also in L_3, for every m.
- No contradiction here.
- So we use the extra condition, making the repeating part appear near the beginning: $|\,u\,v\,| \leq p$.
- This implies that uv must contain only 0s.
- Then u v v w does yield a contradiction: it adds in at least one 0, in the first part only, yielding unequal-length runs of 0s.

Example 3, revisited

- $L_3 = \{ w\,w \mid w \in \{ 0, 1 \}^* \}$ is non-regular.
- Choose $x = 0^p\,1\,0^p\,1$, where p is the number of states.
- Then x can be written as u v w, with $|v| \geq 1$, so that $u\,v^m\,w$ is also in L_3, for every m, and so that $|\,u\,v\,| \leq p$.
- This implies that uv must contain only 0s.
- Then u v v w does yield a contradiction: it adds in at least one 0, in the first part only, yielding unequal-length runs of 0s.

- Note: It was important to pick the right string to pump.
  - E.g., if we chose x = 010101…, an even number of repetitions of 01, then we could pump all we want and not get a contradiction.
  - The PL might give us x = u v w with v = 0101.
  - Adding in 0101 any number of times yields a string in L_3 .

More Examples

Example 4: Palindromes

- $L_4 = PAL = \{ w \in \{0,1\}^* \mid w = w^R \}$ is non-regular.
- Suppose there is a DFA for PAL with p states.
- Pick a word x in PAL and pump it to get a contradiction.
- Choose $x = 0^p 1\ 0^p$; clearly x is in PAL
- The Pumping Lemma yields x = u v w, $|v| \geq 1$, $|uv| \leq p$, and $u\ v^m\ w$ in PAL for every m.
- Thus, the pumping part is near the beginning of x.
- Since $|uv| \leq p$, uv consists of 0s only.
- Since $|v| \geq 1$, v contains at least one 0.
- Then u v v w must be in PAL.
- But this can't be, because we added at least one 0 in the first part and not in the second part.

Example 5

- L_5 = EQ = { w $\in$ {0,1}* | w contains the same number of 0s and 1s} is non-regular.
- Suppose there is a DFA for EQ with p states.
- Choose $x = 0^p 1^p$ to pump; clearly x is in EQ.
- The Pumping Lemma yields x = u v w, |v| $\geq$ 1, |uv| $\leq$ p, and u v^m w in EQ for every m.
- Since |uv| $\leq$ p, uv consists of 0s only.
- Since |v| $\geq$ 1, v contains at least one 0.
- Then u v v w is supposed to be in EQ, but it isn't.

Example 5

- L_5 = EQ = { w $\in$ {0,1}* | w contains the same number of 0s and 1s} is non-regular.
- Alternative proof:
  - By contradiction.
  - Suppose that EQ is regular.
  - Then EQ $\cap$ 0*1* is also regular. Why?
  - Because 0*1* is regular, and the class of regular languages is closed under intersection.
  - But EQ $\cap$ 0*1* = { 0^n1^n | n $\geq$ 0 } = L_1, which we have already proved is non-regular.
  - Contradiction.

Example 6

- A non-regular unary language, $\Sigma = \{ 1 \}$.
- $L_6 = \{ 1^n \mid n \text{ is a prime number} \}$ is non-regular.
- Suppose L_6 is regular, p = number of states in accepting DFA.
- Let $n \geq p$ be a prime number, choose $x = 1^n$.
- The Pumping Lemma yields $x = u\ v\ w$, $|v| \geq 1$, and $u\ v^m\ w$ in L_6 for every m.
- So we have $x = 1^n = 1^a\ 1^b\ 1^c$, where $u = 1^a$, $v = 1^b$, $w = 1^c$.
- Since $u\ v^m\ w$ in L_6 for every m, we have that every number of the form $n + k\ b$ is prime, for every nonnegative integer k.
- But that can't be true:
  - Consider k = n.
  - Then $n + k\ b = n + n\ b = n\ (1+b)$, which is not prime (since $b \geq 1$).

Example 7: Pumping down

- $L_7 = \{ 0^i 1^j \mid i > j \}$ is non-regular.
- It doesn't work to pump up within the initial block of 0s---wouldn't produce something outside L_7.
- But we can pump down, if we choose the right x.
- Choose $x = 0^{p+1} 1^p$, obviously in L_7.
- Then $x = u\ v\ w$, $|v| \geq 1$, $|uv| \leq p$, and every $u\ v^m\ w$, for any $m \geq 0$, is in L_7.
- Considering $m = 0$, we know that $u\ w$ is in L_7.
- v consists of just 0s, and contains at least one 0.
- So removing v removes at least one 0, which yields a string that is not in L_7.

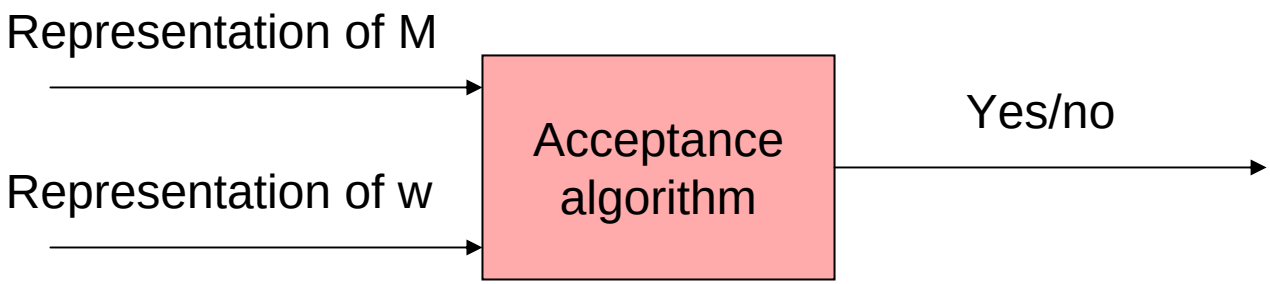
Algorithms that Answer Questions about FAs

Answering questions about FAs

- We can ask general questions about DFAs, NFAs, and regular expressions and try to answer them algorithmically, that is, by procedures that could be programmed in some ordinary programming language.
- Represent the DFAs, etc., by strings in some standard way, e.g., tuples with some encoding of a transition table.
- Sample questions:
  - Acceptance: Does a given DFA M accept a given input string w?
  - Nonemptiness: Does DFA M accept any strings at all?
  - Totality: Does M accept all strings?
  - Nonempty intersection: Do $L(M_1)$ and $L(M_2)$ have any string in common?
  - Subset: Is $L(M_1)$ a subset of $L(M_2)$?
  - Equivalence: Is $L(M_1) = L(M_2)$?
  - Finiteness: Is L(M) a finite set?
  - Optimality: Does M have the smallest number of states for a DFA that recognizes L(M)?

Acceptance problem

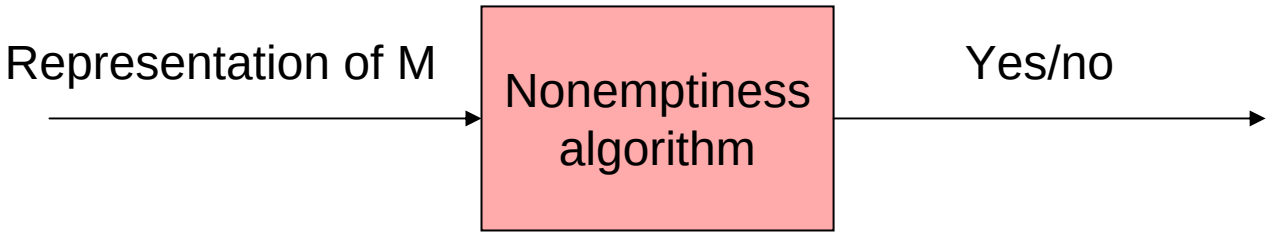
- Does a given DFA M accept a given input string w?



- Need representation for w as well as M, since DFAs have different input alphabets, whereas the program has a fixed alphabet.
- Algorithm:
  - Emulate M on w to see if it ends up at an accepting state.
  - Do the emulation using table lookup for each step.

Nonemptiness problem

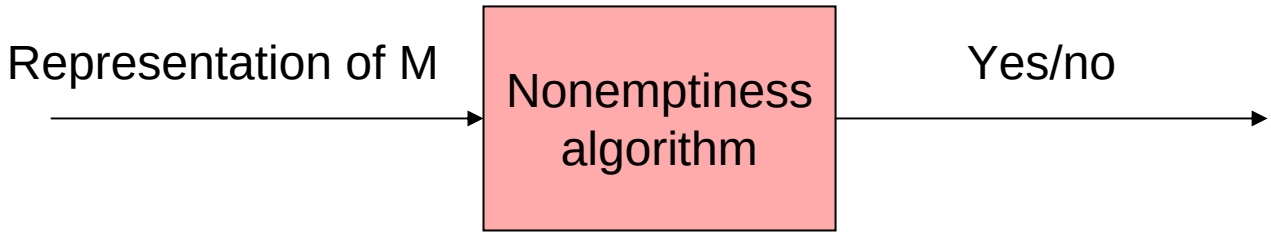
- Does DFA M accept any strings at all?
- That is, is L(M) $\neq \varnothing$?



- Note that L(M) $\neq \varnothing$ if and only if there is a path from the start state q_0 to an accepting state.
- Algorithm 1:
  - Search the DFA digraph from q_0, using some standard search method like BFS or DFS, until you stop finding new states.
  - See if any accepting state has been reached.
- Algorithm 2:
  - If M accepts anything, it accepts some string of length < n, where n is the number of states. Try all these strings.

Nonemptiness problem

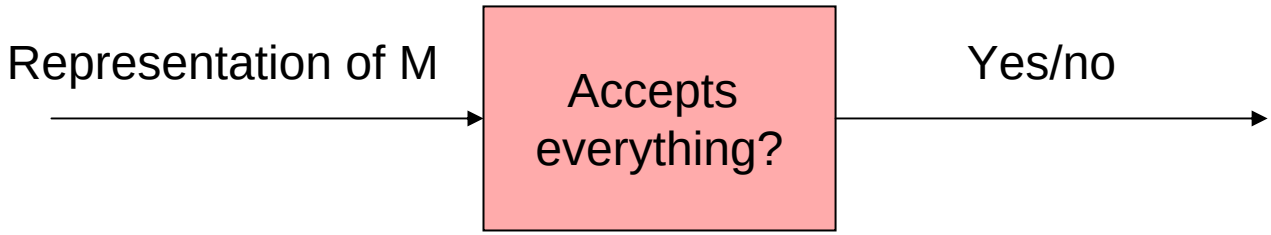
- Does DFA M accept any strings at all?
- That is, is L(M) $\neq \emptyset$?



- Algorithm 2:
  - If M accepts anything, it accepts some string of length < n, where n is the number of states. Try all these strings.
- But why is it true that, if M accepts anything, then it accepts some string of length < n?
  - Otherwise consider the shortest w accepted; must have $|w| \geq n$.
  - Then the states encountered in processing w must repeat somewhere, by Pigeonhole Principle.
  - Short-circuit the intervening segment and get a shorted accepted word, contradicting the assumption that w is shortest.

Totality problem

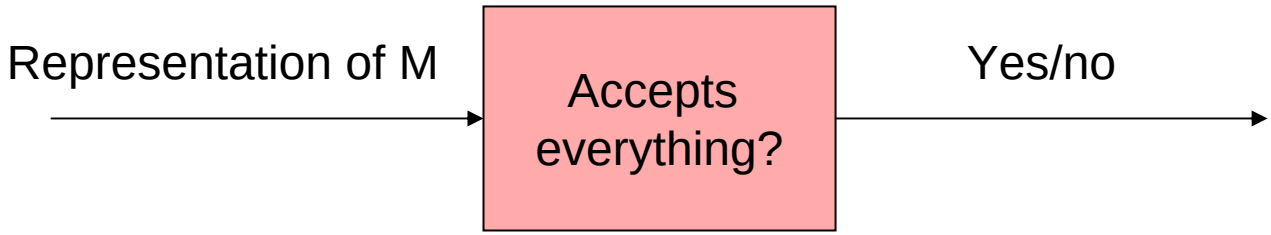
- Does M accept all strings?
- That is, is L(M) = Σ^* ?



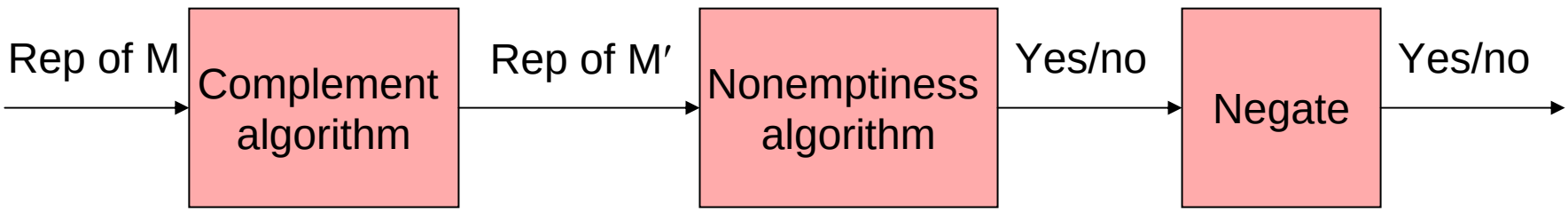
- We can't try all strings…
- Note that L(M) = Σ^* if and only if there is no path from the start state q_0 to a nonaccepting state.
- Algorithm 1:
  - Search to see if there is a path to a nonaccepting state and give the opposite answer.
- Algorithm 2:
  - Transform M into a machine M′ with L(M′) = $(L(M))^c$.
  - Ask if L(M′) is nonempty, and give the opposite answer.

Totality problem

- Does M accept all strings?
- That is, is L(M) = Σ^* ?

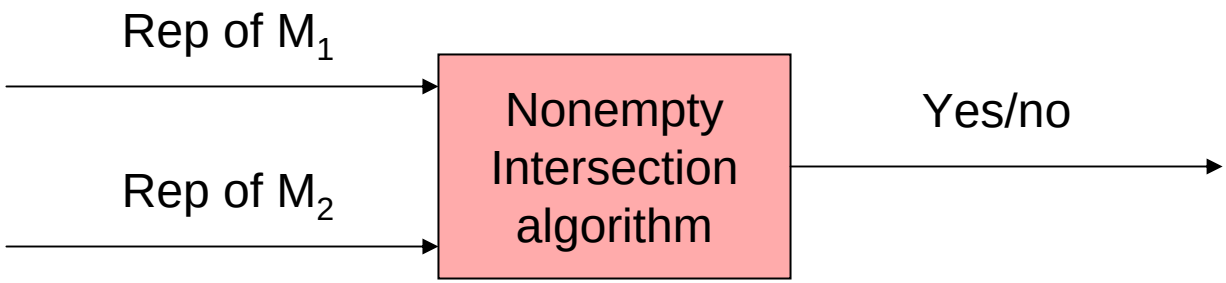


- Algorithm 2:
  - Transform M into a machine M′ with L(M′) = $(L(M))^c$.
  - Ask if L(M′) is nonempty, and give the opposite answer.
  - Both steps can be done with programs.



Nonempty intersection problem

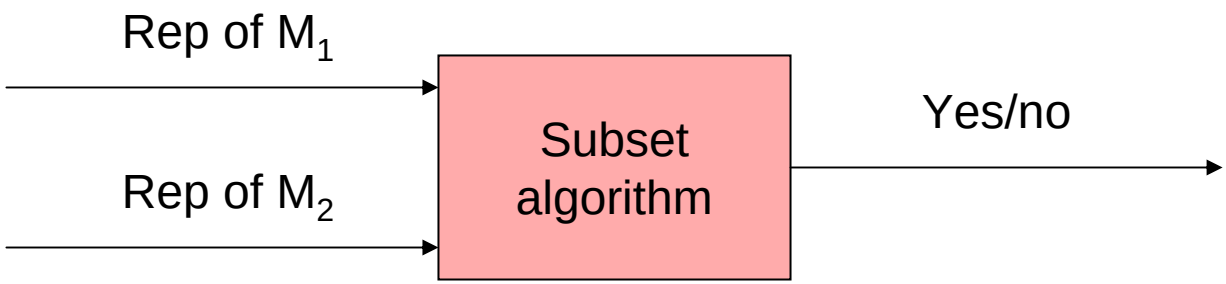
- Do $L(M_1)$ and $L(M_2)$ have any string in common?
- That is, is $L(M_1) \cap L(M_2) \neq \emptyset$?



- Algorithm:
  - Get a DFA M_3 (algorithmically) that recognizes $L(M_1) \cap L(M_2)$.
  - Ask if $L(M_3)$ is nonempty and give the same answer.

Subset problem

- Is $L(M_1) \subseteq L(M_2)$?
- Note that $L(M_1) \subseteq L(M_2)$ if and only if $L(M_1) \cap (L(M_2))^c = \varnothing$.



- Algorithm:
  - Get a DFA M_3 (algorithmically) that recognizes $(L(M_2))^c$.
  - Get another DFA M_4 (algorithmically) that recognizes $L(M_1) \cap (L(M_2))^c$.
  - Ask if $L(M_4)$ is empty and give the same answer.

Equivalence problem

- Is $L(M_1) = L(M_2)$?
- Note that $L(M_1) = L(M_2)$ if and only if both $L(M_1) \subseteq L(M_2)$ and $L(M_2) \subseteq L(M_1)$.

- Algorithm:
  - Test whether $L(M_1) \subseteq L(M_2)$ (algorithmically).
  - Test whether $L(M_2) \subseteq L(M_1)$ (algorithmically).
  - Say yes iff both say yes.

Finiteness problem

- Is L(M) a finite set?
- Can't try all words…
- As for the nonemptiness test, we would like to find a limited range of lengths to test for membership, sufficient to answer the question.
- Pumping Lemma is useful here!
- Claim 1: If M accepts even one string of length $\geq$ n (the number of states), then L(M) is infinite.
  - Because we can pump up that string repeatedly.
- But we can't try all strings of length $\geq$ n…???
- Claim 2: If L(M) is infinite, then M accepts at least one string x with $n \leq |x| < 2n$.
- With these claims, we have an easy algorithm:

Finiteness problem

- Is L(M) a finite set?
- Claim 1: If M accepts even one string of length $\geq$ n (the number of states), then L(M) is infinite.
- Claim 2: If L(M) is infinite, then M accepts at least one string x with $n \leq |x| < 2n$.
- Algorithm (assuming Claims 1 and 2):
  - Try all strings of lengths n,…,2n-1.
  - If any are in L(M), then L(M) is infinite.
    - By Claim 1.
  - If none are in L(M), then L(M) is finite.
    - By Claim 2.

Finiteness problem

- Claim 2: If L(M) is infinite, then M accepts at least one string x with $n \leq |x| < 2n$.
- Proof of Claim 2:
  - Since L(M) is infinite, it includes a string of length $\geq 2n$.
  - Choose one, x, of minimum length $\geq 2n$.
  - Apply the Pumping Lemma to x, writing x = u v w with $|uv| \leq n$ and $|v| \geq 1$.
  - Pumping down, we know that u w is in L(M).
  - Show that $n \leq |uw| < 2n$:
    - $|uw| \geq n$:
      - Because $|uvw| = |x| \geq 2n$, and $|v| \leq |uv| \leq n$.
    - $|uw| < 2n$:
      - Suppose not, so $|uw| \geq 2n$.
      - Impossible because u w is shorter than x and x was chosen to be a minimum length string in L(M) with length $\geq 2n$.

Optimality problem

- Does M have the smallest number of states for a DFA that recognizes L(M)?
- Algorithm 1:
  - Enumerate all DFAs (up to isomorphism) with fewer states than M and test all for equivalence with M.
  - How to enumerate:
    - Use canonical state names q_0, q_1, q_2,…
    - For each fixed number n of states, where n < number of states of M, list all machines with states q_0,…,q_{n-1}.
    - List these by considering all possible collections of arrows, all choices of accept states.
    - Details LTTR.
- Algorithm 2:
  - Apply a state minimization algorithm for DFAs to M (see Sipser, Exercise 7.40).
  - Merges states of M, as far as possible, while maintaining equivalence.
  - Can prove that such an approach in fact yields the minimum number of states.

Questions about NFAs

- Sample questions:
  - Acceptance: Does a given NFA M accept a given input string w?
  - Nonemptiness: Does NFA M accept any strings at all?
  - Totality: Does M accept all strings?
  - Nonempty intersection: Do $L(M_1)$ and $L(M_2)$ have any string in common?
  - Subset: Is $L(M_1)$ a subset of $L(M_2)$?
  - Equivalence: Is $L(M_1) = L(M_2)$?
  - Finiteness: Is L(M) a finite set?
  - Optimality: Does M have the smallest number of states for an NFA that recognizes L(M)?
- Can answer all but the last simply by translating to DFAs and answering the same question.
- Optimality: List and test equivalence.

Questions about regular expressions

- Sample questions:
  - Acceptance: Does the language denoted by a given regular expression R include a given input string w?
  - Nonemptiness: Is $L(R) \neq \emptyset$?
  - Totality: Is $L(R) = \Sigma^*$?
  - Nonempty intersection: Is $L(R_1) \cap L(R_2) \neq \emptyset$?
  - Subset: Is $L(R_1) \subseteq L(R_2)$?
  - Equivalence: Is $L(R_1) = L(R_2)$?
  - Finiteness: Is L(R) a finite set?
  - Optimality: Is R the shortest regular expression whose language is L(R)?
- Can answer all but the last simply by translating to DFAs and answering the same question.
- Optimality: List and test equivalence.

Next time…

- Computability theory
- Readings:
  - Sipser Chapter 3
  - The Nature of Computation, Chapter 8
  - GITCS notes, lecture 4

6.045J / 18.400J Automata, Computability, and Complexity

Spring 2011