Everything Else About Regular Languages

Mark Greenstreet, CpSc 421, Term 1, 2006/07

Lecture Outline

Regular Expressions

- More Pumping Lemma Examples
- Properties of Regular Languages
- Model Checking

One More Pumping Lemma Example

- Let $A = \{w \mid w = w^{\mathcal{R}}\}$ (in English, the palindrome language). Let the input alphabet be $\{\mathsf{a}, \mathsf{b}\}$.
- Let A is not regular.
  - Let p be a proposed pumping lemma length.
  - Let $w = \mathsf{a}^p\,\mathsf{b}\mathsf{a}^p$. $w \in A$.
  - Let $xyz = w$ with $|y| > 0$ and $|xy| \leq p$. Let $n = |xy|$.
  - Then $xy = \mathsf{a}^n$, and $xy^iz = \mathsf{a}^{p+(i-1)|y|}\mathsf{b}\mathsf{a}^p$.
  - If $i \neq 1$, then $xy^iz \notin A$.
  - A does not satisfy the conditions of the pumping lemma.
  - A is not regular.

Remarks About the Pumping Lemma

- If A is finite (i.e. $|A|$ is finite), then A trivially satisfies the pumping lemma. Let

$$p = 1 + \max_{w \in A} |w|$$

  There are no strings in A with length at least p, and the conditions of the pumping lemma are (vacuously) satisfied.
- WARNING: There are non-regular languages that satisfy the pumping lemma. For example,

$$\begin{aligned} \Sigma &= \{\mathsf{a}, \mathsf{b}, \mathsf{c}\} \\ A &= (\mathsf{b} \cup \mathsf{c})^* \mathsf{a}^* \mathsf{b}^n \mathsf{c}^n \end{aligned}$$

  The language A is not regular, but it satisfies the conditions of the pumping lemma.
- Satisfying the conditions of the pumping lemma is a necessary but not sufficient condition for showing that a language is regular.

$(\mathbf{b} \cup \mathbf{c})^*\mathbf{a}^*\mathbf{b}^n\mathbf{c}^n$

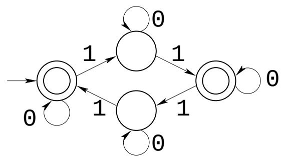
- A is not regular.
  - The regular languages are closed under intersection.
  - The language a*b*c* is regular.
  - Let $A' = A \cap ($a*b*c$^*) =$ a*b^{n}c^n. A' is not regular.
    - Let p be a proposed pumping lemma constant for A'.
    - Let $w =$ b^{p}c^p. Note that $w \in A'$.
    - Pumping w changes the number of b$'s$ in w and produces a string that isn't in A' (even though it is still in A).
    - Therefore, A' is not regular.
  - Therefore A is not regular.

$(\mathbf{b} \cup \mathbf{c})^*\mathbf{a}^*\mathbf{b}^n\mathbf{c}^n$

- A satisfies the pumping lemma.
  - Let $p = 1$. Let w be any non-empty string in A.
  - If w has no a's, then $w \in ($b $\cup$ c$)^+$.
    - Let $x = \epsilon$, $y =$ the first symbol of w, and $z =$ the rest of w.
    - $xyz = w$. $|y| = 1 > 0$. $|xy| = 1 = p$.
    - $xy^iz \in ($b $\cup$ c$)^* \subset A$.
    - The pumping lemma is satisified.
  - If w has one or a's and w starts with an a, then $w \in$ a$^+$b^{n}c^n.
    - Again, let $x = \epsilon$, $y =$ the first symbol of w, and $z =$ the rest of w.
    - $xy^iz \in a^*($b $\cup$ c$)^* \subset A$.
  - If w has one or a's and w starts with a b or c, then $w \in ($b $\cup$ c$)^+$a$^+$b^{n}c^n.
    - Again, let $x = \epsilon$, $y =$ the first symbol of w, and $z =$ the rest of w.
    - $xy^iz \in ($b $\cup$ c$)^*$a$^+$b^{n}c$^n \subset A$.
  - A satisfies the conditions of the pumping lemma.
  - Satisfying the conditions of the pumping lemma is a necessary but not sufficient condition for showing that a language is regular.
  - There are stronger versions of the pumping lemma that A violates. I plan to put a question or two on homework 4 that will explore this.

Distinguishable Strings

- Let A be a language with alphabet Σ.
- Strings x and y are distinguished by A iff there is a string z such that $xz \in A$ and $yz \notin A$ or vice-versa.
- If x and y are not distinguished by A we write $x \equiv_A y$. As suggested by the notation, $\equiv_A$ is an equivalence relation (see Sipser problem 1.51).
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes A.
  If $\delta(q_0, x) = \delta(q_0, y)$ then $x \equiv_A y$.
- A has at most $|Q|$ equivalence classes.



- A language is regular iff it generates a finite number of equivalence classes. (See Sipser problem 1.52)

An Example: $\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k$, $(i = 1) \Rightarrow (j = k)$

- Let $A = \mathtt{a}^i\mathtt{b}^j\mathtt{c}^k$ with $i, j, k \in \mathbb{N}$ and if $i = 1$ then $j = k$ (from Sipser problem 1.54).
- A is not regular.
  - For any $m \in \mathbb{N}$, let $x_m = \mathtt{ab}^m$.
  - For any $m, n \in \mathbb{N}$ with $m \neq n$, x_m and x_n are distinguishable:
    $x_m\mathtt{c}^m \in A$ and $x_n\mathtt{c}^m \notin A$.
  - A generates an infinite number of equivalence classes.
  - A is not regular.
- A similar argument shows that $(\mathtt{b} \cup \mathtt{c})^*\mathtt{a}^*\mathtt{b}^n\mathtt{c}^n$ is not regular.
  For each $m \in \mathbb{N}$, $x_m = \mathtt{bab}^m$ is in a different equivalence class.

Language Emptiness

- Let’s say we have a regular language specified by giving a DFA, NFA, or regular expression.
- We can convert NFAs and REs to DFAs; so, I’ll assume that we have a DFA.
- Is this language empty?
  - Construct the transition graph for the DFA.
  - Use any graph exploration algorithm to find a path from the start state to an accepting state.
  - If it finds a path, then the language is non-empty.
  - If there is no path, then the language is empty.
- If you use breadth-first search, then you find a shortest string in the language.
- How would you test whether or not the language of a DFA, NFA, or RE is complete (i.e. Σ^*)?

DFA Equivalence

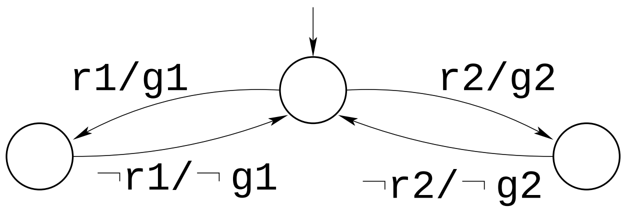
- Let $M_1 = (Q_1, \Sigma, \delta_1, q_{0,1}, F_1)$ and $M_1 = (Q_1, \Sigma, \delta_1, q_{0,1}, F_1)$.
- We say that M_1 and M_2 are equivalent iff $L(M_1) = L(M_2)$.
- To test for language equivalence, we note that we can construct a DFA, $M_1 \oplus M_2$ that accepts iff exactly one of M_1 or M_2 accept ($\oplus$ indicates “exclusive-OR”).
- M_1 and M_2 are equivalent iff $L(M_1 \oplus M_2)$ is empty.

DFA Minimization

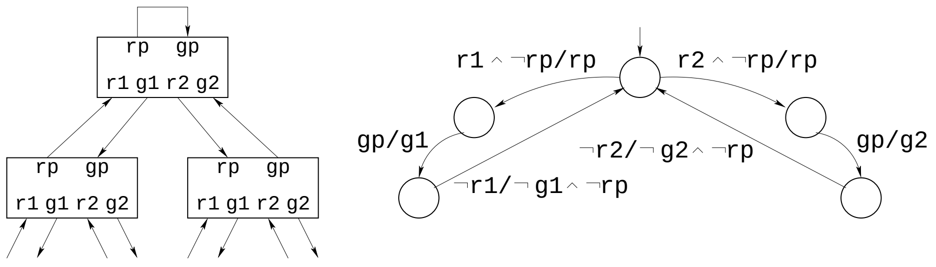
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Let $q_i, q_j \in Q$.
- Let $M' = (Q, \Sigma, \delta, q_0, \{q_i\})$ be a DFA.
  If $L(M') = \emptyset$, then there is no input string that takes M to state q_i.
  We can remove q_i from Q (and δ) without changing $L(M)$.
- Let $M'_i = (Q, \Sigma, \delta, q_i, F)$ and $M'_j = (Q, \Sigma, \delta, q_j, F)$.
  If $L(M'_i) = L(M'_j)$ then states strings that lead to q_i and strings that lead to q_j are indistinguishable by $L(M)$. We can
  - Replace all arcs into q_j with arcs into q_i.
  - Eliminate q_j.
- When all states are reachable from q_0 and distinguishable, we have a DFA with $|Q|$ equal to the number of equivalence classes of $L(M)$. This is the smallest DFA that recognizes M.
- The smallest DFA is unique up to the names for the states.

Model Checking

- Mutual Exclusion:
  - Two clients:



  - Multiple clients:



  - The third version appears at the end of the notes (slide 18).

Can You Really Do This?

- YES
- The algorithms used in practice are more optimized than the simple versions given here.
- The big challenge is the exponential increase in the number of states when going from NFAs to DFAs (and similar operations). This is called the “state-explosion” problem.
  - For some problems, explicitly keeping track of the set of states is practical.
  - Symbolic techniques work very well in many cases: represent the set by a predicate that identifies the members of a set. Boolean satisifiability checkers can handle very large problems.
  - There’s no “silver-bullet” that works for all problems.
  - This is an area of active research.

Other Finite Automata Topics

- Automata on infinite strings, e.g. Büchi Automata.
  A string is accepted if the machine makes an infinite number of visits of accepting states.
- Timed automata: Give upper and lower bounds on how long the machine can remain in a state. Use this to prove responsiveness and other timing properties of systems modeled by finite automata.
- Quantum Finite Automata:
  - Machine state is a quantum state. It can be the quantum superposition of two or more base states.
  - This gives the machine a limited, probabilistic, form of non-determinism.
  - The transition relation must be reversible.
- 2DFAs: The machine can move move its read head either left or right with each state transition. 2DFAs recognize the regular languages (and nothing else).
- Homomorphisms: a whole other set of closure properties that involve mapping strings in one alphabet to strings in another. The regular languages are closed under homomorphisms and inverse homomorphisms.

Proving Regularity

- ● Show a DFA, NFA, or regular expression for A.
- ● Find zero or more regular langauges, $B_1, \ldots, B_k$ that can be combined using the regular operators (union, complement, concatenation, asteration) to produce A. Note that union and complement mean you can make any boolean combination (e.g. AND, exclusive-OR, ...).
- ○ Show that A has a finite number of equivalence classes of distinguished strings (see slide **??**). Note that I just touched on this approach lightly in this lecture. It's not an "official" part of the course material. So, I won't give you problems that require using this method, but you notice a problem for which this provides an easy solution, in which case, you can feel free to use it.

Proving Non-Regularity

- ● Use the pumping lemma.
- ● Find zero or more regular langauges, $B_1, \ldots, B_k$ such tht you can combine A with these languages using the regular operators to produce a language that is clearly not regular (see slide **??**).
- ○ Assume that A is regular, and prove a contradiction.
- ○ Show that A has an infinite number of equivalence classes of distinguished strings.
- ● Most (all) problems that you'll get in this course can be handled by the first two methods. As with proving regularity, I won't give you any problems that require using the equivalence classes method, but now that you know it, you might find that it you can use it to get simpler solutions.

Proving Closure Properties

- Typically, you get asked a question of the form:

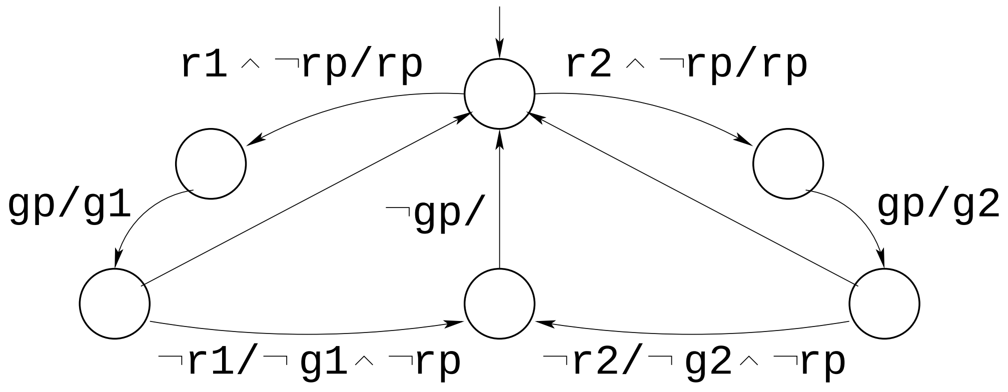
  Show that if A is regular then some particular variation on A is regular (e.g. the regular languages are closed under complement and asteration).

  or

  Show that if $A_1 \ldots A_k$ are regular then some particular way of creating a new language from them creates a regular language (e.g. the regular languates are closed under union and concatenation).

- Let A' be the new language that gets created. Because we want to show that A' is regular, we can use the methods from slide 15.
- Quite frequently, we'll construct a NFA for A'. Note that becasue $A_1 \ldots A_k$ are assumed to be regular, we can assume that we are given NFAs (or DFAs, or REs) for those languages. It is often very helpful to use pieces of these NFAs, their states, their transition relation, etc., to define an NFA for A'.

The Last Mutual Exclusion Design


r1 ∧ ¬rp/rp
r2 ∧ ¬rp/rp
gp/g1
¬gp/
gp/g2
¬r1/¬ g1 ∧ ¬rp
¬r2/¬ g2 ∧ ¬rp