Regular Languages II

Intro to Finite Automata

First things

- Homework #1 due today
- Speaking of homework
  - Homework #2 (Due Sept 24)
    - 3.1a,b
    - 3.9a,d,e
    - 3.17d,e
    - 3.19a,d,e
    - 3.33a,b,c (Use subset construction)

Questions

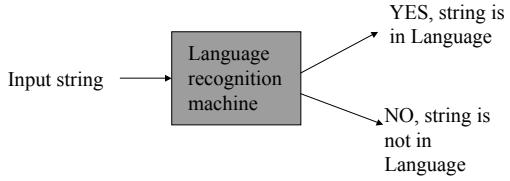
- Any questions before we start?

Regular Languages

- Today we start looking at our first class of languages: Regular languages
  - Means of defining: Regular Expressions
  - Machine for accepting: Finite Automata
- We looked at regular expressions, now lets look at finite automata.

String Recognition machine

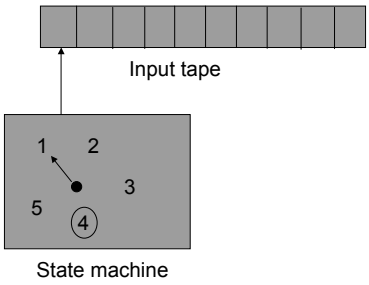
- Given a string and a definition of a language (set of strings), is the string a member of the language?



Finite Automata

- A finite automaton(FA) consists of
  - A read tape
  - A machine that can be in one or more "states"
    - Start state – The state the machine is in at the beginning of execution
    - Accepting states – The state(s) the machine has to be in after execution in order for a string to be "accepted"

Finite Automata



Finite Automata

- How the automaton works
  - Read a character on the tape
  - Based on the character read and the current "state" of the machine, put your machine into another "state"
  - Move the read head to the right
  - Repeat the above until all characters have been read.

Finite Automata

- Testing a string for membership
  - Place the string to be tested on the read tape
  - Place the machine into the start "state"
  - Let the machine run to completion
  - If, upon completion, the machine is in an accepting "state", the string is accepted, otherwise it is not.

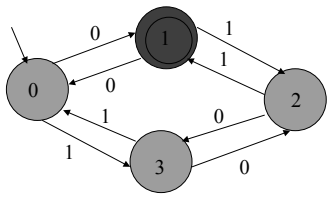
Finite Automata

- Transition function
  - Defines what state the machine will move into given:
    - The current machine state
    - The character read off the tape
  - Sometimes illustrated as directed graph where nodes represent states and edges represent transitions.

Finite Automata

- $L = \{x \in \{0,1\}^* \mid x$ contains an odd number of 0s and an even number of 1s$\}$

	# 0s	#1s
0	even	even
1	odd	even
2	odd	odd
3	even	odd



Finite Automata

- This transition can also be given by a table.

state \ character	0	1
0	1	3
1	0	2
2	3	1
3	2	0

Finite Automata Visualization

- JFLAP
  - Java Formal Language and Automata Package
  - By Susan Rodger at Duke University
  - http://www.cs.duke.edu/~rodger/tools/jflap

Finite Automata

- Demo using FLAP

Finite Automata <-> RE

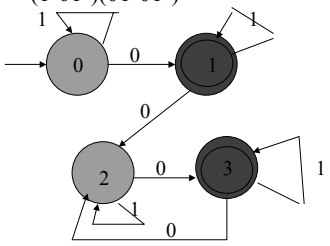
- In coming weeks we will formally prove that:
  - Every language L accepted by a Finite Automaton there is a regular expression that describes L.
  - For every language L that can be expressed by a regular expression there is a finite automaton that will accept only strings from L
- For now we shall give some intuitive examples

Finite Automata from a RE

- Example
  - L = represented by $(1^*01^*)(01^*01^*)^*$

Finite Automata from a RE

- A FA that accepts L
  - $(1^*01^*)(01^*01^*)^*$



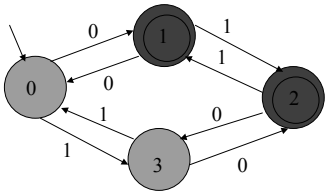
Finite Automata from a RE

- Let's see that in action

Finite Automata from a RE

$(1^*01^*)(01^*01^*)^*$ is the regular expression for $L = \{x \in \{0,1\}^* \mid x$ contains an odd number of 0s $\}$

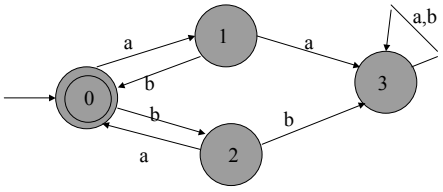
	# 0s	#1s
0	even	even
1	odd	even
2	odd	odd
3	even	odd



RE from a FA

- Intuitive notions
  - A branch in an FA implies a union
  - Sequence from one state to the next implies concatenation
  - A loop in the graph implies a Kleene star.
  - Recall that for a string to be accepted, the machine must end up in an accepting state.

RE from a FA



$(ab + ba)^*$

RE from a FA

- Observation
  - If the start state is also an accepting state then Λ is a member of the language accepted by the FA.
  - Questions?
  - Now let's get Mathematical about things.

Finite Automata

- Consists of
  - A set of states
  - A start state
  - A set of accepting states
  - Read symbols
  - Transition function
- Let's define an automata formally

Finite Automata

- A finite automaton (finite-state machine) is a 5-tuple $(Q, \Sigma, q_o, \delta, A)$ where
  - Q is a finite set (of states)
  - Σ is a finite alphabet of symbols
  - $q_o \in Q$ is the start state
  - $A \subseteq Q$ is the set of accepting states
  - δ is a function from Q x Σ to Q (transition function)

Finite Automata

- The transition function
  - δ is a function from Q x Σ to Q
  - $\delta(q, a) = q'$ where
    - $q, q' \in Q$
    - $a \in \Sigma$
  - δ defines, given a current state q and reading character a, to which state the FA will move.

Finite Automata

- Applying the transition function to a string.
  - δ^* is a function from Q x Σ^* to Q
  - $\delta^*(q, x) = q'$ where
    - $q, q' \in Q$
    - $x \in \Sigma^*$
  - δ^* defines, given a current state q and reading a string x, to which state the FA will move once all characters of x are read.

Finite Automata

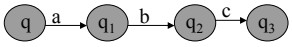
- Recursive definition of δ^*
  - Let M = $(Q, \Sigma, q_o, \delta, A)$ be an FA.
  - Define δ^*: Q x $\Sigma^* \rightarrow Q$
    1. For any $q \in Q$, $\delta^*(q, \Lambda) = q$
    2. For any $y \in \Sigma^*$, $a \in \Sigma$, and $q \in Q$
       $\delta^*(q, ya) = \delta(\delta^*(q, y), a)$

Finite Automata

- Example of δ^*

$\delta^*(q, \Lambda) = q$

$\delta^*(q, ya) = \delta(\delta^*(q, y), a)$



$\delta^*(q, \underline{abc}) = \delta(\delta^*(q,\underline{ab}), c)$
$= \delta(\delta(\delta^*(q, \underline{a}), b), c)$
$= \delta(\delta(\delta^*(q, \underline{\Lambda a}), b), c)$
$= \delta(\delta(\delta(\underline{\delta^*(q, \Lambda),} a), b), c)$
$= \delta(\delta(\underline{\delta(q, a),} b), c)$
$= \delta(\underline{\delta(q_1, b),} c)$
$= \underline{\delta(q_2, c)}$
$= q_3$

Finite Automata

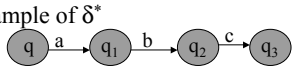
- Another Recursive definition of δ^*
  - Let M = $(Q, \Sigma, q_o, \delta, A)$ be an FA.
  - Define δ^*: Q x $\Sigma^* \rightarrow Q$
    1. For any $q \in Q$, $\delta^*(q, \Lambda) = q$
    2. For any $y \in \Sigma^*$, $a \in \Sigma$, and $q \in Q$
       $\delta^*(q, ay) = \delta^*(\delta(q, a), y)$

Finite Automata

- Example of δ^*

$\delta^*(q, \Lambda) = q$

$\delta^*(q, ay) = \delta^*(\delta(q, a), y)$



$\delta^*(q, \underline{abc}) = \delta^*(\delta(q,a), bc)$
$= \delta^*(q_1, bc)$
$= \delta^*(\delta(q,b), c)$
$= \delta^*(q_2, c)$
$= \delta^*(q_2, c\ \underline{\Lambda})$
$= \delta^*(\delta(q,c), \Lambda)$
$= \delta^*(q_3, \Lambda)$
$= q_3$

Language accepted by an FA

- Let M = $(Q, \Sigma, q_o, \delta, A)$ be an FA.
- A string $x \in \Sigma^*$ is accepted by M if
  - $\delta^*(q_0, x) \in A$
  - In other words,
    - Starting in your start state q_0
    - Run the machine with input x
    - The machine ends up in an accepting state
  - If a string x is not accepted by M it is said to be rejected by M.

Language accepted by an FA

- The language accepted or recognized by M is:
  - $L(M) = \{ x \in \Sigma^* \mid x \text{ is accepted by M} \}$
- If L is a language over Σ, L is accepted by M iff L = L(M).
  - For all $x \in L$, x is accepted by M.
  - For all $x \notin L$, x is rejected by M.

Kleene Theorem

- A language L over Σ is regular iff there exists an FA that accepts L.
  1. If L is regular there exists an FA M such that L = L(M)
  2. For any FA, M, L(M) is regular
     L(M), the language accepted by the FA can be expressed as a regular expression.

     For now, we will accept this without proof . We will prove it formally in the weeks to come.

Reality Check

- What we've learned so far
  - An FA can be expressed by $(Q, \Sigma, q_o, \delta, A)$
  - Transition function: $\delta: Q \times \Sigma \rightarrow Q$
  - Transition function on strings
    - $\delta^*: Q \times \Sigma^* \rightarrow Q$
    - Defined recursively
  - FA accepting a string
  - The language accepted by an FA
  - Kleene Theorem

Reality Check

- Questions?
- Let's take a break.

Closure Properties of Regular Languages

- In this lecture we will consider the set of operations on which the set of regular languages are closed
  - I.e. if L is a regular language, then Op(L) is also a regular language.

Closure Properties of Regular Languages

- By definition, regular languages are closed under
  - Union
  - Concatenation
  - Kleene Star
- But did you know that they are also closed under
  - Intersection
  - Difference
  - Complement

Closure Properties of Regular Languages

- I.e. if L_1 and L_2 are regular languages then
  1. $L_1 \cup L_2$ is regular
  2. $L_1 L_2$ is regular
  3. L_1^* is regular
  4. $L_1 \cap L_2$ is regular
  5. $L_1 - L_2$ is regular
  6. L_1' is regular

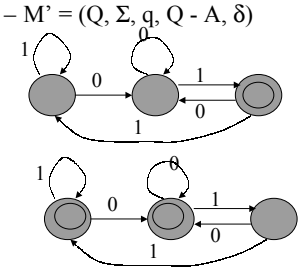
  We will show 1, 4, 5, 6 by constructing a FA that accepts these set operations.

Complement

- Let L be a regular language
  - There exists an FA M= (Q, Σ, q, A, δ) such that L(M) = L.
  - $x \in L$ iff $\delta^*(q, x) \in A$
  - $x \notin L$ iff $\delta^*(q, x) \notin A$
  - $L' = \{x \mid x \notin L\}$
  - So to construct an FA M' that accepts L', we simply let all accepting states in M be non-accepting in M' and all non-accepting states in M be accepting in M'

Complement

- In other words
  - M' = (Q, Σ, q, Q - A, δ)



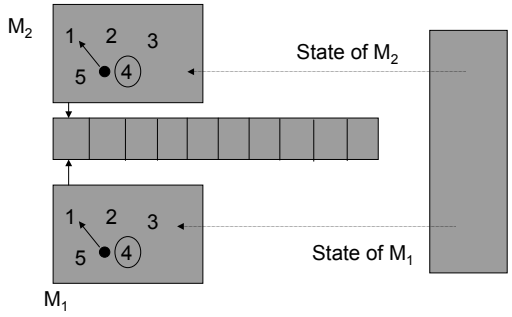
Union, Difference, Intersection

- Now let's try Union, Difference, and Intersection
- Constructive Proof
  - Will build a FA that accepts these languages

Union, Difference, Intersection

- Basic idea
  - If L_1 and L_2 are regular, then by Kleene Theorem, there exists FAs , M_1 and M_2 such that
    - $L_1 = L(M_1)$
    - $L_2 = L(M_2)$
  - We will build an FA, M that, for a single input x, will keep track of the current states of both M_1 and M_2 as they read x.

Union, Difference, Intersection



Union, Difference, Intersection

- Let's be a bit more formal.
  - Let L_1 and L_2 be regular languages and let
  - $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ be an FA that accepts L_1 and
  - $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$ be an FA that accepts L_2

Union, Difference, Intersection

- We will build a new FA, M such that
  - The states of M are an ordered pair (p, q) where $p \in Q_1$ and $q \in Q_2$
  - Informally, the states of M will represent the current states of M_1 and M_2 at each simultaneous move of the machines.

Union, Difference, Intersection

- Formally…
  - $M = (Q, \Sigma, q_0, A, \delta)$ where
    - $Q = Q_1 \times Q_2$
    - $q_o = (q_1, q_2)$
    - $\delta: (Q_1 \times Q_2) \times \Sigma \rightarrow (Q_1 \times Q_2)$
      - $\delta((p,q), a) = (p',q')$ where $p, p' \in Q_1$ and $q, q' \in Q_2$
      - $\delta((p,q), a) = (\delta_1(p,a), \delta_2(q,a))$
    - A will depend on the operation being performed.

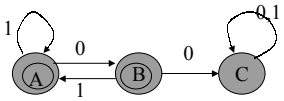
Union, Difference, Intersection

- Set of accepting states
  - Union
    - $A = \{(p,q) \mid p \in A_1 \text{ or } q \in A_2\}$
  - Intersection
    - $A = \{(p,q) \mid p \in A_1 \text{ and } q \in A_2\}$
  - Difference
    - $A = \{(p,q) \mid p \in A_1 \text{ and } q \notin A_2\}$

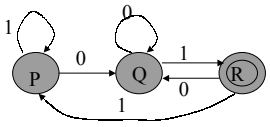
Union, Difference, Intersection

- Let's step through an example
  - Example 3.18 in the book.
  - $L_1 = \{ x \mid 00 \text{ is not a substring of } x\}$
  - $L_2 = \{ x \mid x \text{ ends in } 01\}$

Union, Difference, Intersection



M_1



M_2

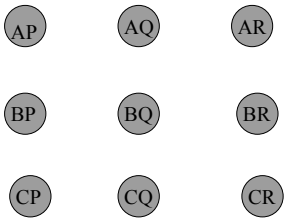
Union, Difference, Intersection

- $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$
  - $Q_1 = \{A, B, C\}$
  - $q_1 = A$
  - $A_1 = \{A, B\}$
- $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$
  - $Q_2 = \{P, Q, R\}$
  - $q_2 = P$
  - $A_2 = \{R\}$

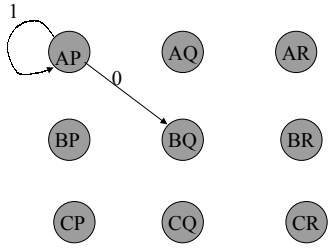
Union, Difference, Intersection

- $M = (Q, \Sigma, q_0, A, \delta)$
  - $Q = \{AP, AQ, AR, BP, BQ, BR, CP, CQ, CR\}$
  - $q_0 = AP$
  - A, set of accepting states, will depend upon the operation for which the FA is being built.

Union, Difference, Intersection



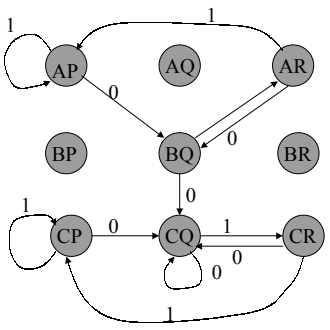
Union, Difference, Intersection



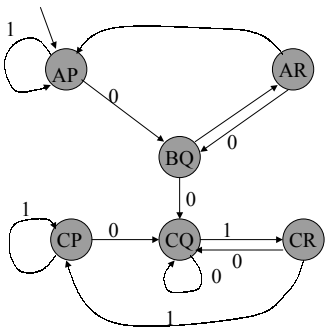
$\delta((A,P), 1) = (\delta_1 (A,1), \delta_2 (P,1)) = (A, P)$

$\delta((A,P), 0) = (\delta_1 (A,0), \delta_2 (P,0)) = (B, Q)$

Union, Difference, Intersection



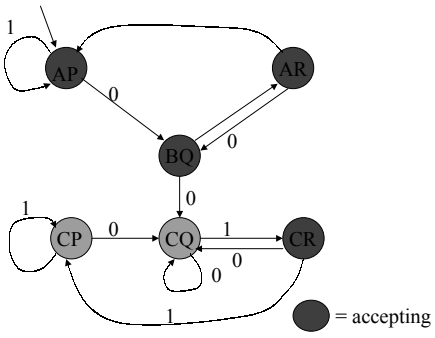
Union, Difference, Intersection



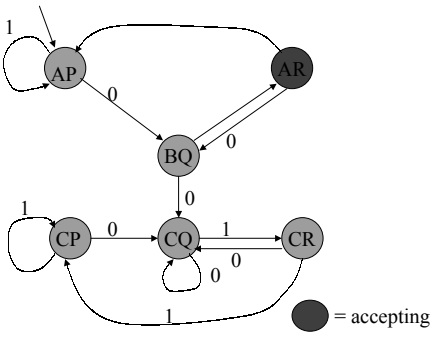
Union, Difference, Intersection

- Finally we can define A, the set of accepting states in M
  - Union
    - $A = \{(p,q) \mid p \in A_1 \text{ or } q \in A_2\}$
  - Intersection
    - $A = \{(p,q) \mid p \in A_1 \text{ and } q \in A_2\}$
  - Difference
    - $A = \{(p,q) \mid p \in A_1 \text{ and } q \notin A_2\}$

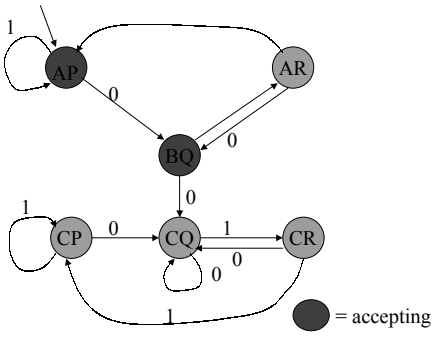
Union



Intersection



Difference



What we have done?

- We've shown regular languages to be closed under Union, Complement, Difference, and Intersection
  - Assumed that Kleene Theorum is true.
    - $L_1 = L(M_1)$ and $L_2 = L(M_2)$
  - Built a FA, M that accepts $L_1 \cup L_2$, L', $L_1 - L_2$ and $L_1 \cap L_2$
    - M simulates the simultaneous running of M_1 and M_2 on the same string.
    - Defined accepting states of M based on the operation.
    - Since M accepts, by Kleene Theorem, each op is a regular language.

Wrapping it all up

- Finite Automata
  - Intuitive Notions
  - Formal Definition
  - Demo using JFLAP
  - Union, Intersection, Complement, Differences of Regular Language
    - Subset construction algorithm

Next Time

- Start Kleene Theorem
  - Introduce non-determinism to Fas.
- Questions?