CMSC330

Finite Automata

Last lecture

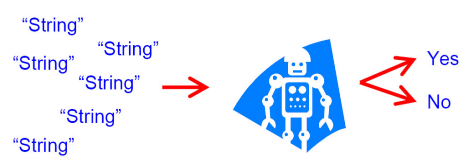
- Languages
  - Sets of strings
  - Operations on languages
- Regular expressions
  - Constants
  - Operators
  - Precedence

This lecture

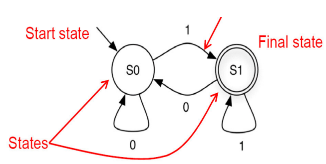
- Finite automata
  - States
  - Transitions
  - Examples
- Types
  - Deterministic (DFA)
  - Non-deterministic (NFA)

Implementing Regular Expressions

- We can implement a regular expression by turning it into a finite automaton
  - A “machine” for recognizing a regular language



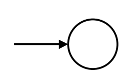
Finite Automata



- Machine starts in start or initial state
- Repeat until the end of the string is reached
  - Scan the next symbol s of the string
  - Take transition edge labeled with s
- String is accepted if automaton is in final state when end of string reached

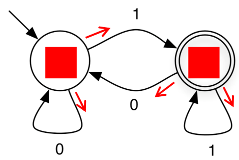
Finite Automata: States

- Start state
  - State with incoming transition from no other state
  - Can have only 1 start state



- Final state
  - State with double circle
  - Can have 0 or more final states

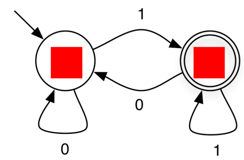
Finite Automaton: Example 1



0 0 1 0 1 1

accepted

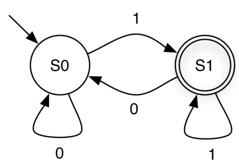
Finite Automaton: Example 2



0 0 1 0 1 0

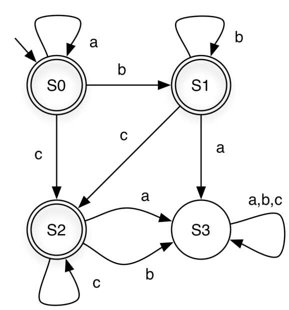
not accepted

What Language is This?



- All strings over {0, 1} that end in 1
- What is a regular expression for this language?
  (0|1)*1

Finite Automaton: Example 3



string	state at end	accepts?
aabcc	S2	Y
acc	S2	Y
bbc	S2	Y
aabbb	S1	Y
aa	S0	Y
ε	S0	Y
acba	S3	N

(a,b,c notation shorthand for three self loops)

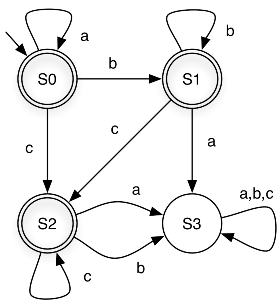
Last Lecture

- Regular Expression Theory
- Introduction to finite state machines

This lecture

- Finite state machines continued

Finite Automaton: Example 3

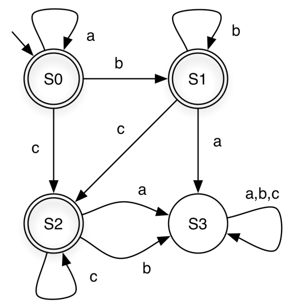


string	state at end	accept s?
aabcc	S2	Y
acc	S2	Y
bbc	S2	Y
aabbb	S1	Y
aa	S0	Y
ε	S0	Y
acba	S3	N

(a,b,c notation shorthand for three self loops)

13

Finite Automaton: Example 3 (cont.)



What language does this DFA accept?

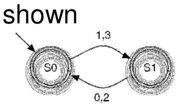
a*b*c*

S3 is a dead state – a nonfinal state with no transition to another state

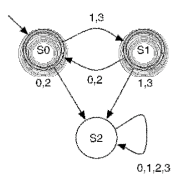
14

Dead State: Shorthand Notation

- If a transition is omitted, assume it goes to a dead state that is not shown



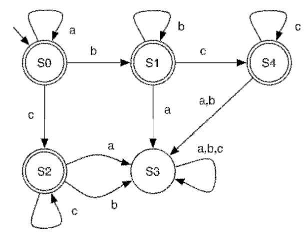
is short for



- Language?
  - Strings over {0,1,2,3} with alternating even and odd digits, beginning with odd digit

15

What Lang. Does This FA Accept?



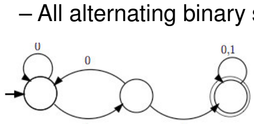
a*b*c* again, so DFAs are not unique

16

Practice

Give the English descriptions and the DFA or regular expression of the following languages:

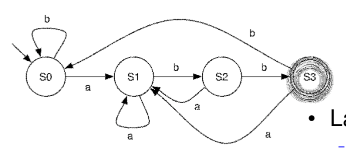
- ((0|1)(0|1)(0|1)(0|1)(0|1))*
  - All strings with length a multiple of 5
- (01)*|(10)*|(01)*0|(10)*1
  - All alternating binary strings



All binary strings containing the substring “11”

17

Finite Automaton: Example 4



- Language?
  - (a|b)*abb

- Description for each state
  - S0 = “Haven’t seen anything yet” OR “seen zero or more b’s” OR “Last symbol seen was a b”
  - S1 = “Last symbol seen was an a”
  - S2 = “Last two symbols seen were ab”
  - S3 = “Last three symbols seen were abb”

18

Practice

- Give the regular expressions and finite automata for the following languages
  - You and your neighbors' names
  - All binary strings containing an even length substring of all 1's
  - All binary strings containing exactly two 1's
  - All binary strings that start and end with the same number

Types of Finite Automata

- Deterministic Finite Automata (DFA)
  - Exactly one sequence of steps for each string
  - All examples so far
- Nondeterministic Finite Automata (NFA)
  - May have many sequences of steps for each string
  - More compact

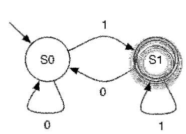
Math Alert!

Formal Definition

- A deterministic finite automaton *(DFA)* is a 5-tuple $(\Sigma, Q, q_0, F, \delta)$ where
  - Σ is an alphabet
    - the strings recognized by the DFA are over this set
  - Q is a nonempty set of states
  - $q_0 \in Q$ is the start state
  - $F \subseteq Q$ is the set of final states
    - How many can there be?
  - $\delta : Q \times \Sigma \times Q$ specifies the DFA's transitions
- A DFA accepts s if it stops at a final state on s (after consuming all chars in s)

Formal Definition: Example

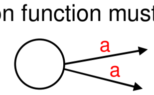
- 5-tuple $(\Sigma, Q, q_0, F, \delta)$
- $\Sigma = \{0, 1\}$
- $Q = \{S0, S1\}$
- $q_0 = S0$
- $F = \{S1\}$



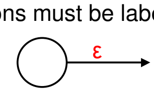
δ	0	1
S0	S0	S1
S1	S0	S1

DFA Requirements

- Can not have more than one transition leaving a state on the same symbol
  - I.e., transition function must be a valid function



- Can not have transitions with empty labels
  - Transitions must be labeled by alphabet symbols



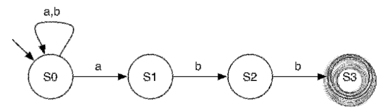
ε-transition

- NFAs do not have these requirements!
  - DFA is a special case of NFA

Nondeterministic Finite Automata (NFA)

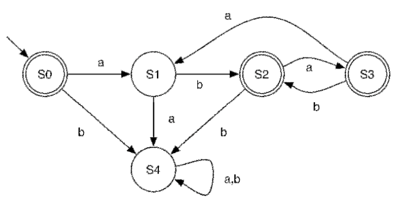
- An *NFA* is a 5-tuple $(\Sigma, Q, q_0, F, \delta)$ where
  - Σ is an alphabet
  - Q is a nonempty set of states
  - $q_0 \in Q$ is the start state
  - $F \subseteq Q$ is the set of final states
  - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ specifies the NFA's transitions
    - Transitions on ε are allowed – can optionally take these transitions without consuming any input
    - Can have more than one transition for a given state and symbol
- An NFA accepts s if there is at least one path from its start to final state on s

NFA for (a|b)*abb



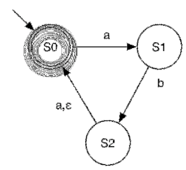
- ba
  - Has paths to either S0 or S1
  - Neither is final, so rejected
- babaabb
  - Has paths to different states
  - One leads to S3, so accepted

Another example DFA



- Language?
  - (ab|aba)*

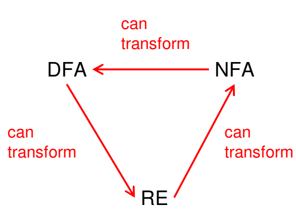
NFA for (ab|aba)*



- aba
  - Has paths to states S0, S1
- ababa
  - Has paths to S0, S1
  - Need to use ε-transition

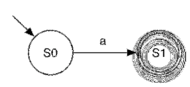
Relating REs to DFAs and NFAs

- Regular expressions, NFAs, and DFAs accept the same languages!



Reducing Regular Expressions to NFAs

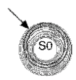
- Goal: Given regular expression e, construct NFA: $\langle e \rangle = (\Sigma, Q, q_0, F, \delta)$
  - Remember regular expressions are defined recursively from primitive RE languages
  - Invariant: $|F| = 1$ in our NFAs
    - Recall F = set of final states
- Base case: a



$\langle a \rangle = (\{a\}, \{S0, S1\}, S0, \{S1\}, \{(S0, a, S1)\})$

Reduction (cont.)

- Base case: ε



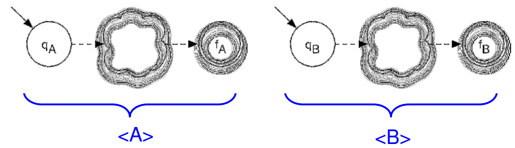
$<ε> = (ε, \{S0\}, S0, \{S0\}, Ø)$

- Base case: Ø

$<Ø> = (Ø, \{S0, S1\}, S0, \{S1\}, Ø)$

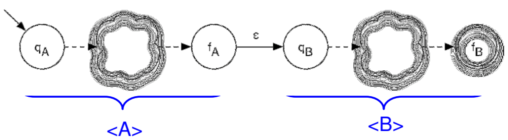
Reduction: Concatenation

- Induction: AB



Reduction: Concatenation (cont.)

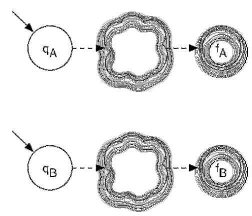
- Induction: AB



- $<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $<B> = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$
- $<AB> = (\Sigma_A\Sigma_B, Q_A U\ Q_B, q_A, \{f_B\}, \delta_A U\ \delta_B U\{(f_A, \varepsilon, q_B)\}\ )$

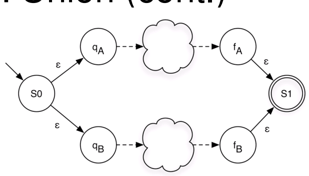
Reduction: Union

- Induction: (A|B)



Reduction: Union (cont.)

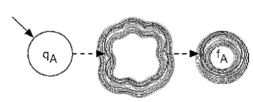
- Induction: (A|B)



- $<A> = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $<B> = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$
- $<(A|B)> = (\Sigma_A\Sigma_B, Q_AQ_B\{S0,S1\}, S0, \{S1\},$
  $\delta_A\ \delta_B U\{(S0,\varepsilon,q_A), (S0,\varepsilon,q_B), (f_A,\varepsilon,S1),$
  $(f_B,\varepsilon,S1)\})$

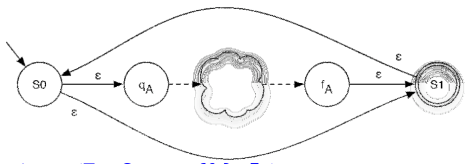
Reduction: Closure

- Induction: A*



Reduction: Closure (cont.)

- Induction: A*



  - <A> = $(\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
  - <A*> = $(\Sigma_A, Q_A \cup \{S0,S1\}, S0, \{S1\}, \delta_A \cup \{(f_A,\varepsilon,S1), (S0,\varepsilon,q_A), (S0,\varepsilon,S1), (S1,\varepsilon,S0)\})$

Reduction Complexity

- Given a regular expression A of size n...

  Size = # of symbols + # of operations

- How many states does <A> have?
  - 2 added for each |, 2 added for each *
  - O(n)
  - That's pretty good!

Practice

- Draw NFAs for the following regular expressions and languages
  - (0|1)*110*
  - 101*|111
  - all binary strings ending in 1 (odd numbers)
  - all alphabetic strings which come after "hello" in alphabetic order
  - (ab*c|d*a|ab)d

Summary

- Finite automata
  - Deterministic (DFA)
  - Non-deterministic (NFA)

- Questions
  - How are DFAs and NFAs different?
  - When does an NFA accept a string?
  - How to convert regular expression to an NFA?