Ling/CSE 472: Introduction to Computational Linguistics

4/1/15

Regular expressions & Finite State Automata

Overview

- Formal definition of regular languages
- Regular expression examples
- Extension to regular expressions
- Substitution in Python
- Finite state automata
- Recognition with FSA -- pseudocode
- DFSA v. NFSA
- Reading questions

Announcement

- Assignment 1 is posted
- Elizalike.py (version 1) due 4/6 at 10am
- Complete assignment due 4/10 at 5pm

Book vs. Lecture

- J&M talk about
  - regular expressions in the context of search,
  - then FSAs,
  - then regular languages
- We'll do
  - regular languages,
  - then regular expressions describing regular languages (with an aside about search)
  - then FSAs

Formal languages

- From the point of view of formal languages theory, a language is a set of strings defined over some alphabet.
- Chomsky hierarchy: a description of classes of formal languages:
  - regular < context-free < context-sensitive < all languages
- Languages from a single level in the hierarchy can be described in terms of the same formal devices.

Three views on the same object

Regular languages can be described by regular expressions and by finite-state automata.

- Regular language: a set of strings
- Regular expression: an expression from a certain formal language which describes a regular language
- Finite-state automaton: a simple computing machine which accepts or generates a regular language

Formal definition of regular languages: symbols

- ϵ is the empty string
- ø is the empty set
- Σ is an alphabet (set of symbols)
- Examples of alphabets:
    - Σ = {a,b,!}
    - Σ = {c,a,t,d,o,g,f,x}
    - Σ = {cat, dog, fox}
    - Σ = {1, 3, 5, 7}

Formal definition of regular languages

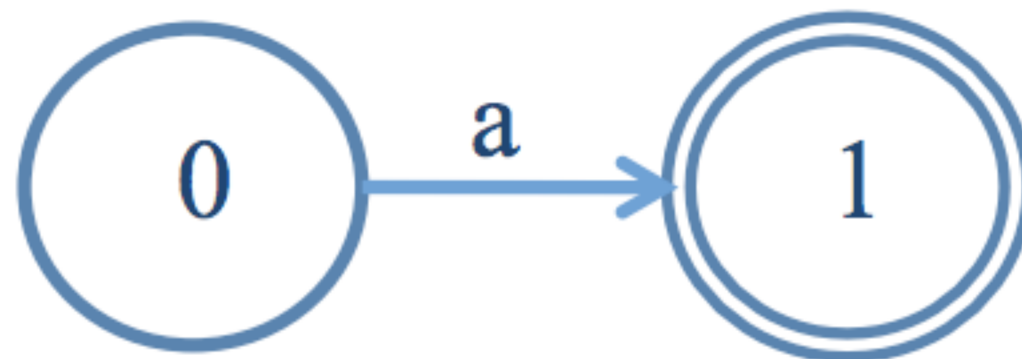
The class of regular languages over Σ is formally defined as:

- $\emptyset$ is a regular language
- $\forall a \in \Sigma \cup \epsilon$, $\{a\}$ is a regular language
- If L1 and L2 are regular languages, then so are:
  - $L1 \cdot L2 = \{xy \mid x \in L1, y \in L2\}$ (concatenation)
  - $L1 \cup L2$ (union or disjunction)
  - $L1^*$ (Kleene closure)

(Jurafsky & Martin 2009:39)

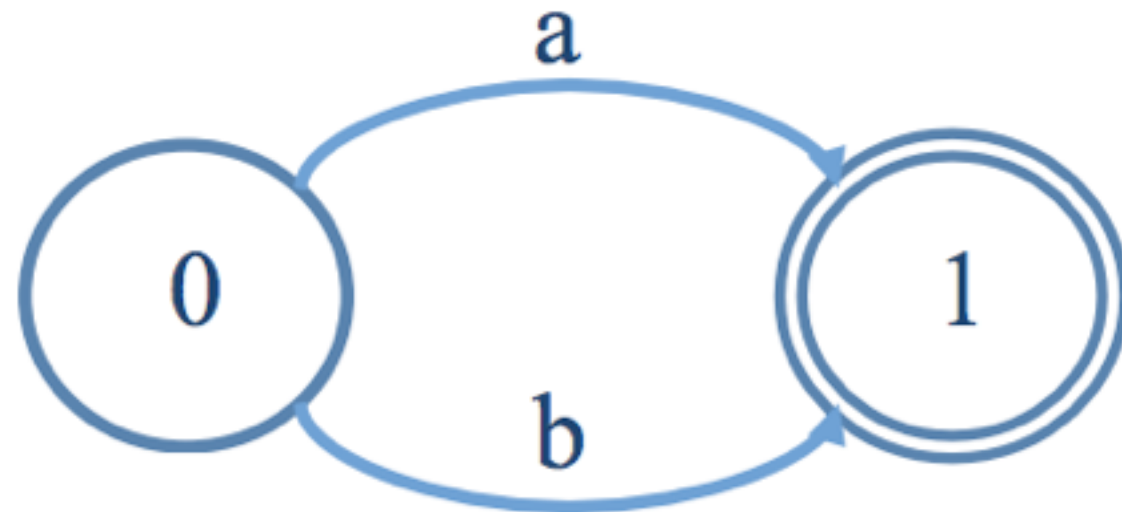
Examples: Single Character

- Alphabet: $\Sigma = \{a\}$
- Language (set of strings): $L = \{a\}$
- Regular expression: /a/
- Finite state machine:



Examples: Disjunction/union

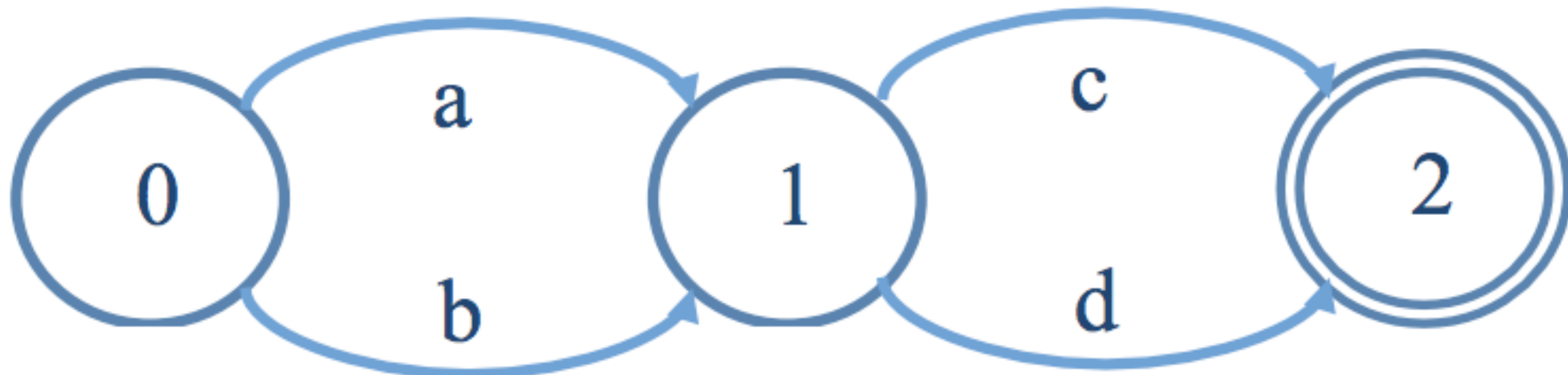
- Alphabet: $\Sigma = \{a, b\}$
- Langauge (set of strings): {a, b}
- Regular expression: /(a|b)/
- Finite state machine:



- This language/regexp/FSA also reflects the union of two languages: $\{a\} \cup \{b\}$

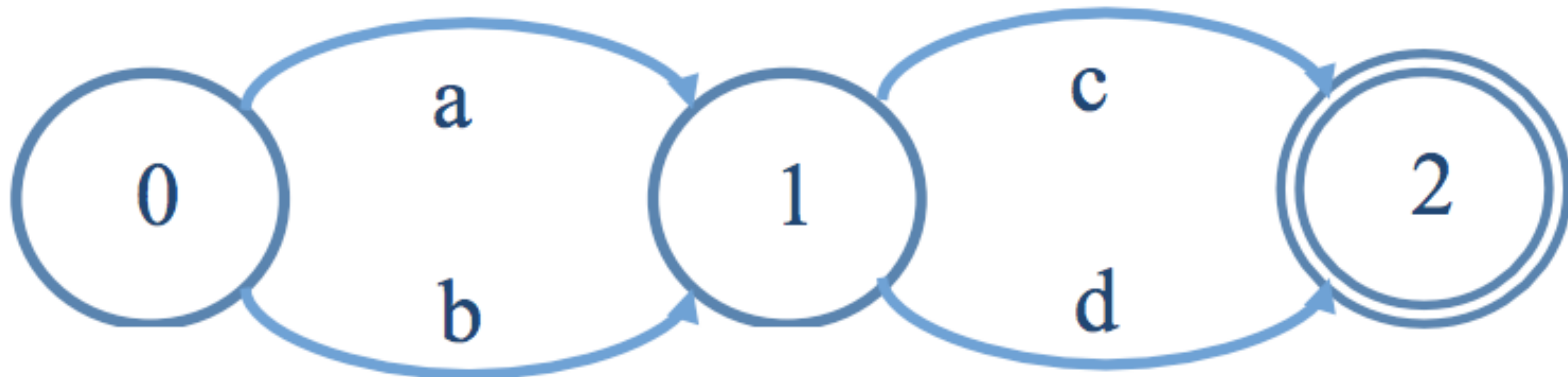
Examples: Concatenation

- Σ = {a, b, c, d}
- L = {ac, ad, bc, bd}
- Regular expression: /(a|b)(c|d)/
- Finite state machine:



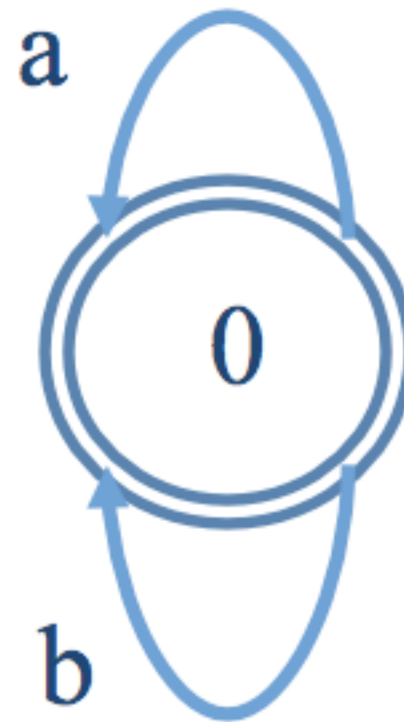
Examples: Concatenation

- Note: This language/regexp/FSA also reflects the concatenation of two languages:
    - L1 = {a, b} · L2 = {c, d}



Examples: Kleene closure

- Σ = {a, b} (again)
- L = {ϵ, a, b, aa, bb, ab, ba, bb, ...}
- Regular expression: /(a|b)*/
- Finite state machine:



Regular expression examples

- abc
- a|bc
- (a|bb)c
- a[bc]
- a*b
- a?b
- [^a]*th[aeiou]+[a-z]*
- ^a
- ^a.*z$
- \bthe\b

Regular expressions: An extension

- Parentheses allow you to ‘save’ part of a string and access it again.
- In general:
  - ( ) creates a group that can be referenced
  - *n* refers back to the *n*th group in a regular expression
- For example, to search for repeated words: /([a-z]+)\1 \1/
- That will find any group of one or more lower case alphabetic characters repeated twice (and separated by a space): ‘I called kitty kitty kitty.’

Regular expressions: An extension

- Note: This extension to Python/Perl/grep/MS “regular expression” syntax actually takes them beyond the realm of regular expressions in the Chomsky hierarchy sense.

- The languages generated by “regular expressions” augmented with this kind of memory device are NOT regular languages, i.e., cannot be recognized by FSAs.

- Why?

So what are regular expressions good for?

So what are regular expressions good for?

- Search

So what are regular expressions good for?

- Search
- Search and replace

So what are regular expressions good for?

- Search
- Search and replace
- Morphotactics: Defining classes of words as sequences of morphemes

So what are regular expressions good for?

- Search
- Search and replace
- Morphotactics: Defining classes of words as sequences of morphemes
- Chunk parsing: Finding units within running text that satisfy patterns such as D A* N

Regular expression substitution in Python

- import re
- re.sub(pattern, replacement, string); use r to mark pattern as a regex
- \g<n> references the nth group

```
>>input = ‘The Kiwis is a team.’
>>input =
re.sub(r’(\W)is(\W)’,’\g<1>are\g<2>’,input)
>>print input
‘The Kiwis are a team.’
```

Back to FSAs

- An abstract computing machine
- Consists of a set of states (or nodes in a directed graph) and a set of transitions (labeled arcs in the graph)
- Three kinds of states: plain, start, final

FSA as transition table

- An FSA can also be represented as a transition table:

	Input		
State	a	b	c
0	1	3	-
1:	1	2	3
2:	-	3	-
3:	-	-	-

Recognizing a regular language

FSAs can be used to recognize a regular language.

- Take the FSA and a “tape” with the string to be recognized.
- Start with the start of the tape and the FSA’s start state.
- For each symbol on the tape, attempt to take the corresponding transition in the machine.
- If no transition is possible: reject.
- When the string is finished, check whether the current state is a final state.
- Yes: accept. No: reject.

Notes on pseudocode

- Some basic components of algorithms:
  - Loops (e.g., while x is true; for every x in …)
  - Conditionals (e.g., if … do this, else do this)
  - Variable assignment (e.g., i = i+1)
  - Evaluating expressions (e.g. if x = 2, print x + 3)
  - Input values

D-RECOGNIZE in pseudo-code

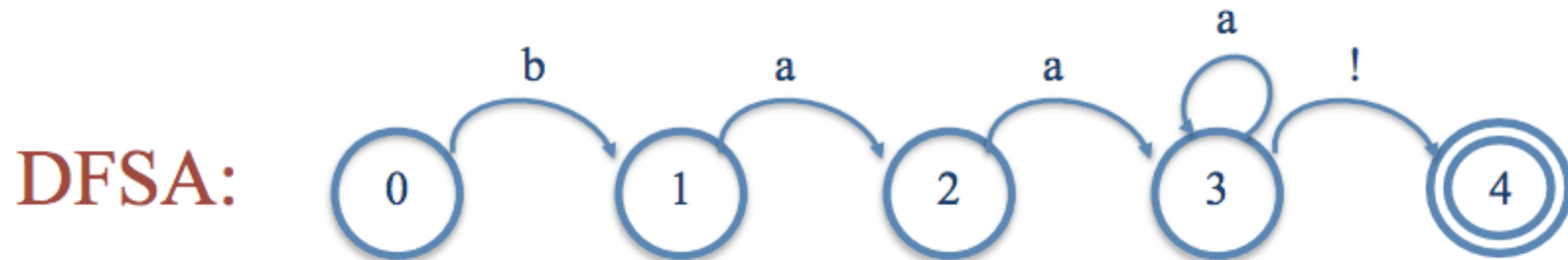
```
function D-Recognize(tape, transition-table) returns accept or reject
    index ← beginning of tape
    current-state ← initial state on transition-table
    loop
        if end of tape has been reached then
            if current-state is an accept state then
                return accept
            else
                return reject
        elsif transition-table[current-state, tape[index]] is empty then
            return reject
        else
            current-state ← transition-table[current-state, tape[index]]
            index ← index + 1
    end
```

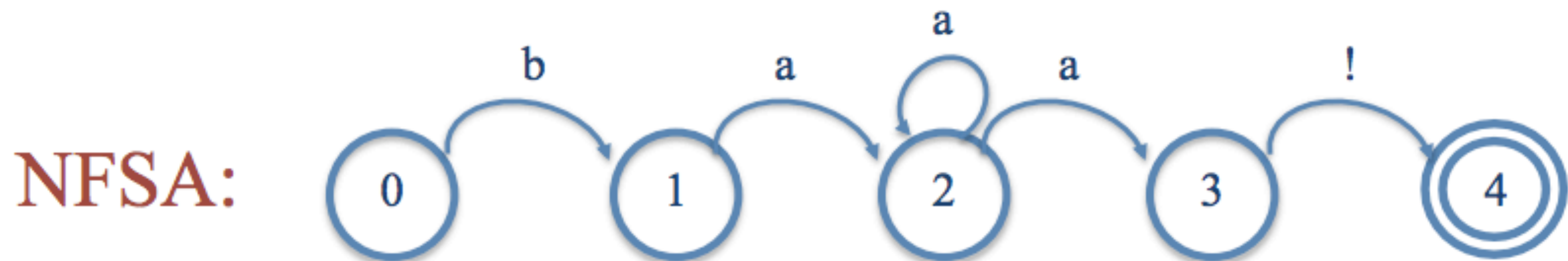
Note: This is a slightly modified version of J&M, Figure 2.12, pg29

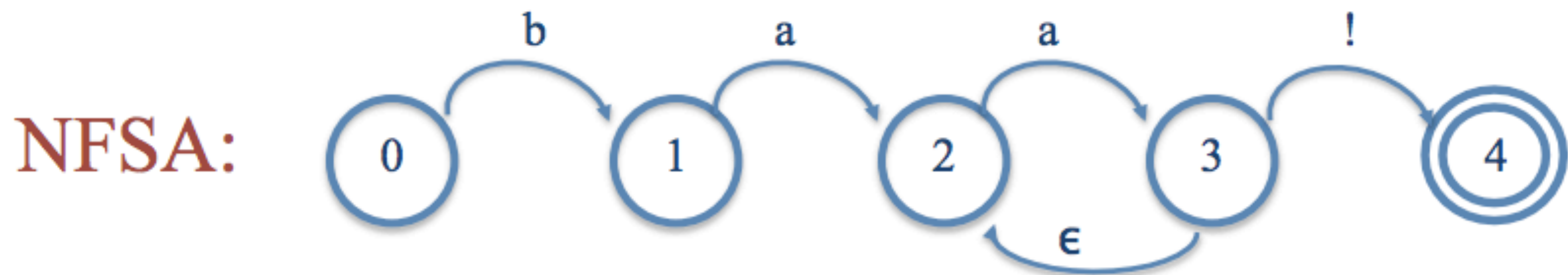
Non-deterministic FSA (NFSA)

- The FSAs considered so far are deterministic (DFSA): there is only one choice at each node.
- NFSAs include more than one choice at (at least) one node.
- Those choices might include ϵ-transitions, i.e., unlabeled arcs that allow one to jump from one node to another without reading any input.
- Note: Any NFSA can be rewritten as a DFSA.

Equivalent (D)FSA and NFSA







Problem with Non-determinism

- Non-determinism can lead to dead ends when recognizing strings. How?

- Types of solutions:
  - backup
  - look-ahead
  - parallelism

- The backup technique is essentially to record a search-state at each decision point consisting of the position on the input tape and the state (node) of the machine. Then paths to dead ends can be retraced and different decisions made.

- What's the use/point of NFSAs, if there's always an equivalent DFSA?

Overview

- Formal definition of regular languages
- Regular expression examples
- Extension to regular expressions
- Substitution in Python
- Finite state automata
- Recognition with FSA -- pseudocode
- DFSA v. NFSA
- Reading questions

Reading questions

- If the Kleene * matches with zero or more occurrences wouldn't everything thats not the character or expression before * be an instance of a zero occurrence?

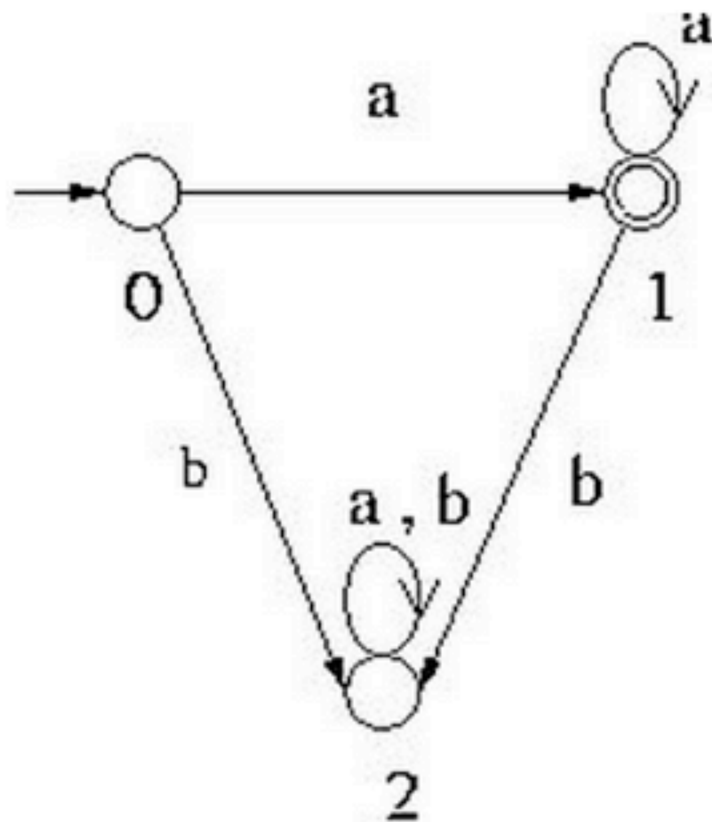
- When referring to regular languages I understand the results of intersection, difference, and complementation, but what does the reversal of a language (L1R) look like or result in?

- Why are regular languages closed under complementation? How would you create a finite state machine that is the complement of another FSM? I sort of understand how you would take the complement of a regular expression (and how that would stay regular), but how would it work with the automata?

From: http://www.cs.odu.edu/~toida/nerzic/390teched/regular/fa/complement.html

Let $M = < Q , \Sigma , q_0 , \delta , A >$ be a DFA that accepts a language L. Then a DFA that accepts the complement of L, i.e. Σ^* - L, can be obtained by swapping its accepting states with its non-accepting states, that is $M_c = < Q , \Sigma , q_0 , \delta , Q - A >$ is a DFA that accepts Σ^* - L .

For example the following DFA accepts the language a^+ over $\Sigma = \{ a , b \}$.



accepts a^+

Reading questions

- What is the function of finite state machines and what are the reasons we use them to represent regular expressions?

- Are FSAs just abstractions? How are they implemented? How can they be used to generate output? How does an NFSA keep track of where it was when it has to back up?

- When is it better to use an NFSA and when is it better to convert to a DFSA?

Reading questions

- How did Regular Expressions manage to maintain such a high level of standardization?
- I found it really interesting that a simple conversation could be created just using regular expressions, and I wonder if this would be possible without using substitutions involving memory. I also wonder if this means that a conversation involving just "small talk" in general is a regular language. It seems to me you could represent it with an FSA...
- Have there been any attempts to make the Turing test more objective? Are there other proposed tests of machine intelligence?

First-day survey comment: Please recommend outside reading material and resources

- ACL Anthology: http://aclweb.org/anthology/
- Morgan Claypool Synthesis Lectures on Human Language Technologies: http://www.morganclaypool.com/loi/hlt
- Recordings from recent (NA)ACL conferences:
  - NAACL 2013: http://techtalks.tv/naacl/2013/
  - ACL 2014: http://techtalks.tv/acl2014/