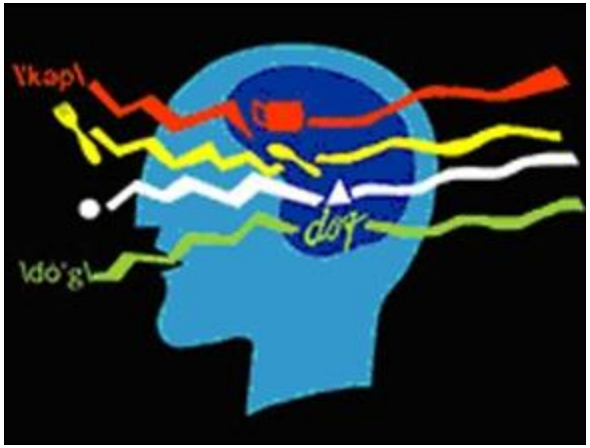
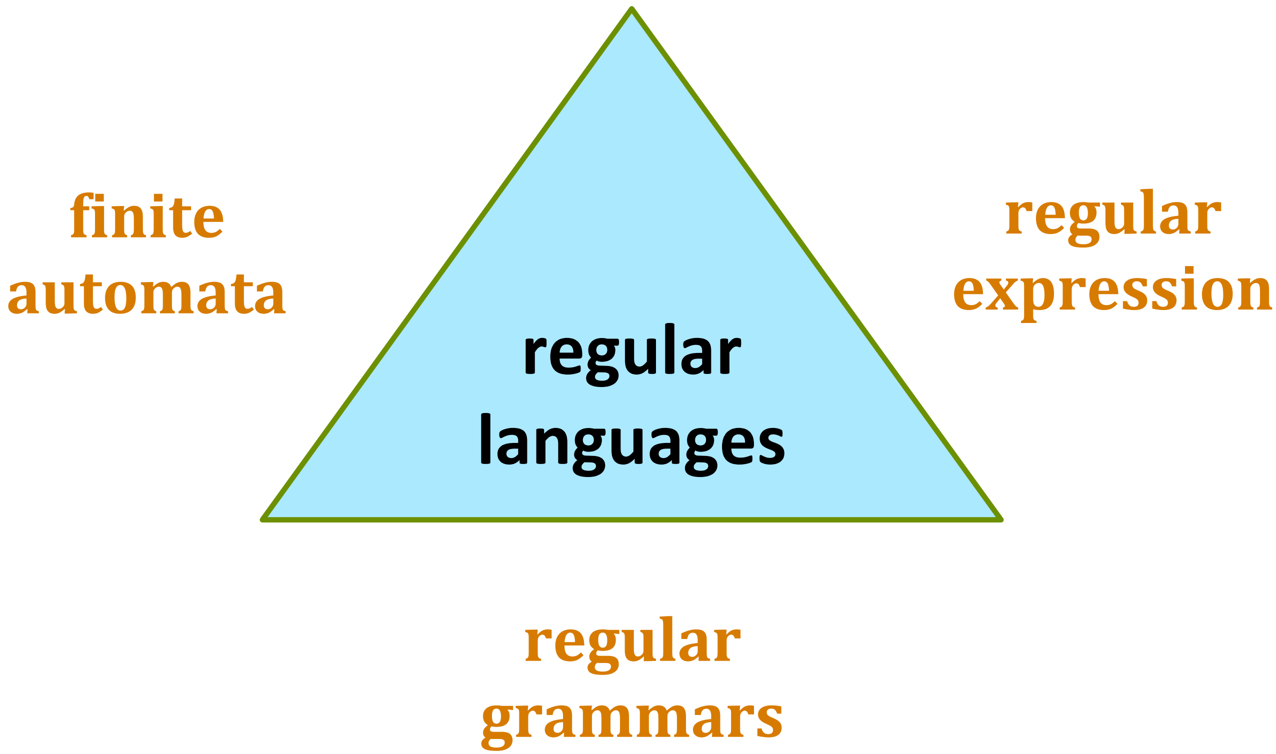
Topic 3 (Pt 1):
Finite State Automata (FSA) and Regular Expressions in NLP



Formal Language Theory

- Three equivalent formal ways to look at regular languages



What is a *regular language?*

- A **regular language** is a **formal language** (a possibly infinite set of finite sequences of symbols from a finite alphabet) that satisfies the following:
  - can be accepted/recognized by a deterministic finite state machine
  - can be accepted/recognized by a non-deterministic finite state machine
  - can be expressed or described using **regular expression**

Example of regular language

- Regular expressions represent regular languages and look like abbreviations for them

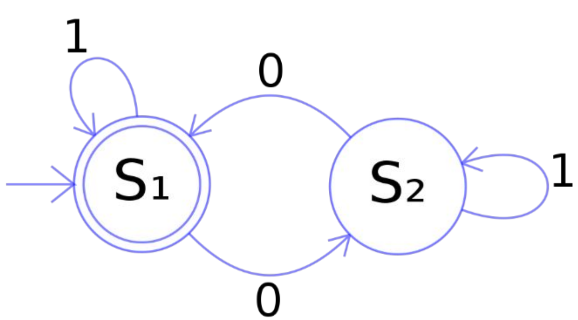
Regular Expression	Corresponding Regular Language
a+bc	{a,bc}
a(b+c)	{ab, ac}
(a+b)(a+c)(L+a)	{aa, ac, ba, bc, aaa, aca, baa, bca}
a*(b+cc)	{b, cc, ab, acc, aab, aacc, aaab, aaacc,...}
a+bb*	{a, b, bb, bbb, bbbb, bbbbb, ...}
(a+bb)*	{L, a, bb, aa, abb, bba, bbbb, aaa, ...}
a*b*	{ L, a, b, aa, ab, bb, aaa, aab, abb, bbb,...}

Set Theory and Regular Languages

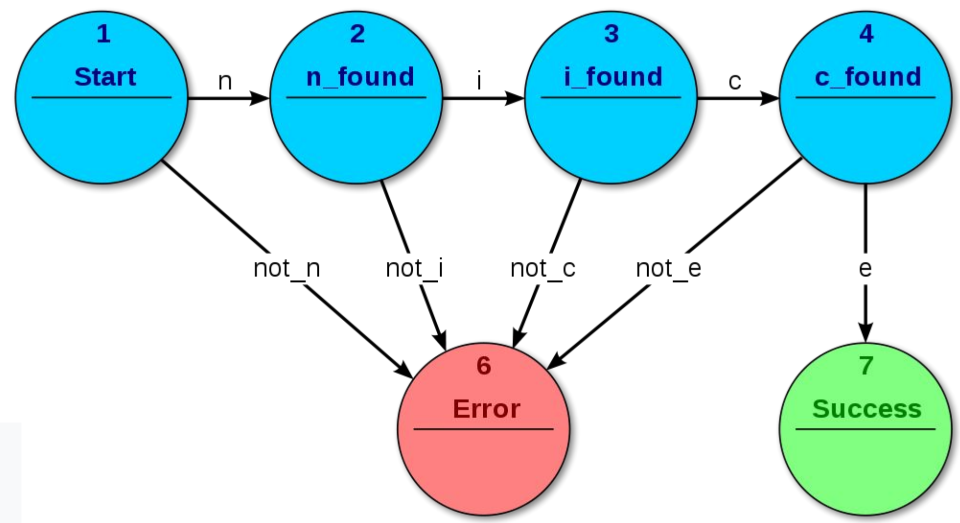
- Sets A set is an abstract, unordered collection of distinct objects, called members or elements of the set. When some element x is a member of a set A, we write $x \in A$ (x is a member of A)
- The set of vowels of the English alphabet is V_E = {a, e, i, o, u } . Its members are a, e, i, o and u.
- The set of articles in German is A_G = {ein, eine, einer, einem, einen, eines, der, die, das, dem, den, des}
- The set of short vowels in Arabic is V_A= {◌ُ,◌َ ,◌ِ }

What is a *Finite State Machine (FSM)?*

- An abstract mathematical model of computation
- Simulate sequential logic



Determines whether a binary number has an even number of 0s, where is an **accepting state**.



Acceptor FSM: parsing the string "nice"

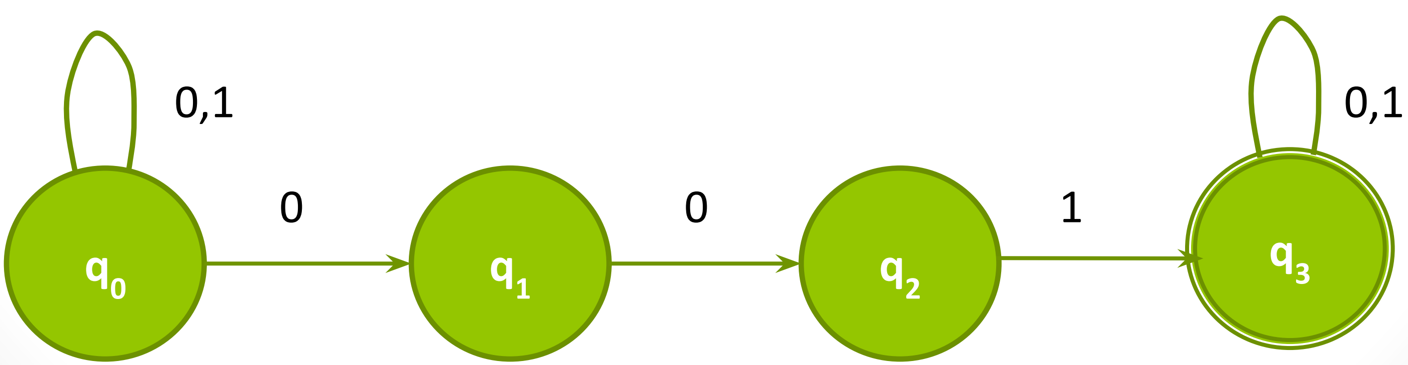
https://en.wikipedia.org/wiki/Finite-state_machine

What is *Finite State Automata (FSA)*?

- It is an abstract machine that can be in exactly one of a finite number of states at any given time.

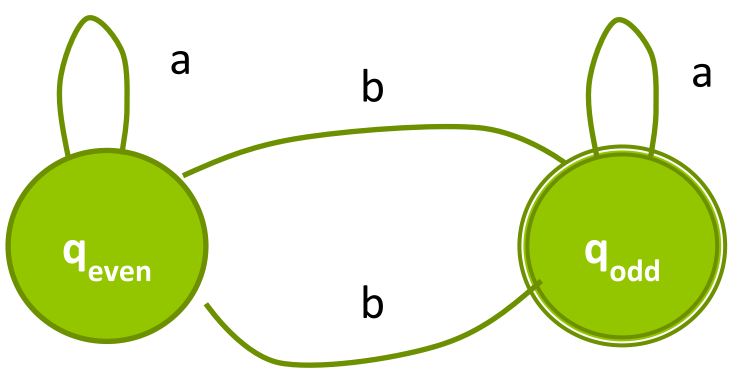
Example for Finite State Automata (FSA)

- Assume the set of symbols/alphabet accepted by the FSA is {0,1}.
- The language generated by the FSA is a string that contains the substring 001
- Example of accepted string/language = {0010,1001,001,1111110011111}
- Example of unaccepted string/language = {11,0000, 10}

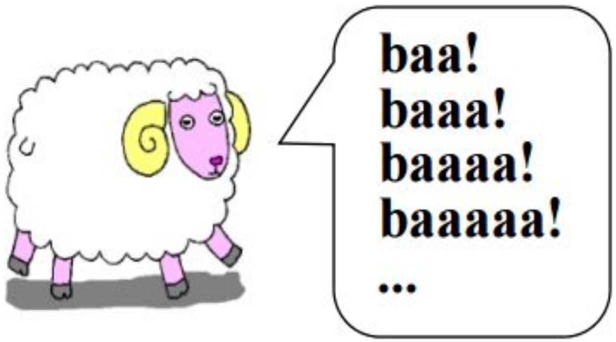


Example for Finite State Automata (FSA)

- Assume the set of symbols/alphabet accepted by the FSA is {a,b}.
- The language generated by the FSA contains odd number of b’s
- Example of accepted string/language = {bbb, ababab, aab, aaabaabbabab}
- Example of unaccepted string/language = {bb,bbaabb}



Finite State Automata (FSA)



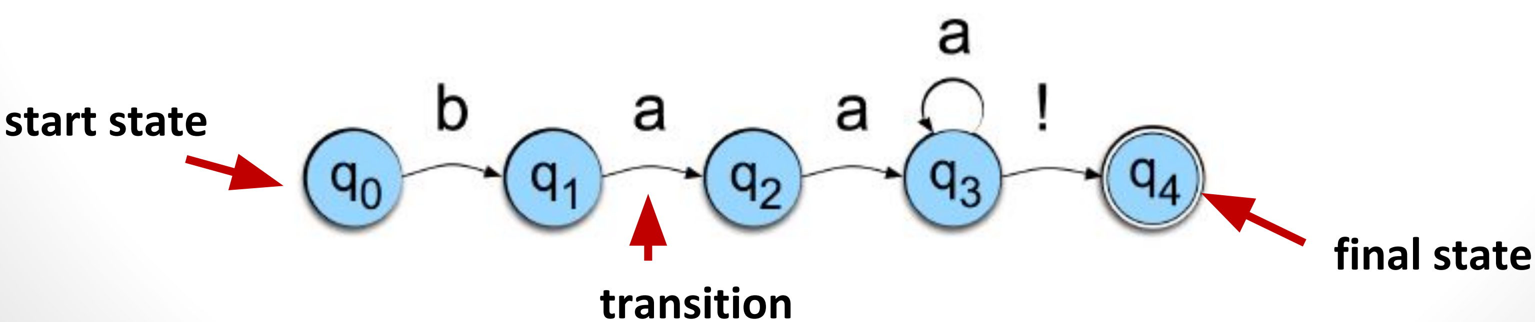
Regular expression

/baa+!/

/^baa+!$/

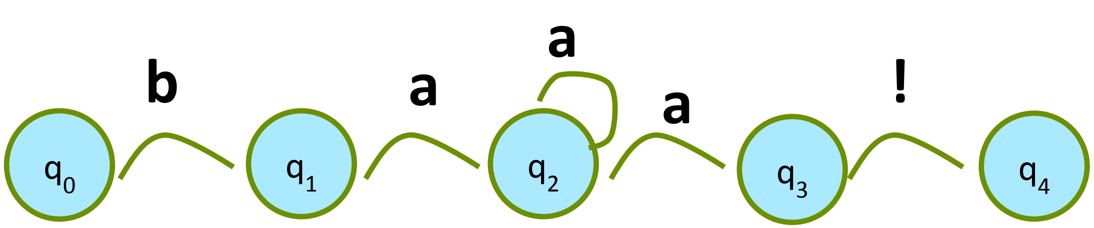
/baaa*!/

- Sheep talk:
  - baa!, baaa!, baaaa!, baaaaa!, ...
- Finite State Automaton



Sheep Finite State Automata

- The FSA has 5 states
- *{b, a, !}* are in its alphabet
- "baa!" is the minimum accepted language
- q_0 is the start state
- q_4 is the accept state
- The FSA has 5 transitions
- Other possible machines?



Formal Finite State Automata

- FSA can be specified as:
  - The set of states, **Q**
  - A finite alphabet **Σ**
  - A start state
  - A set of accept states
  - The transition function that maps **Q x Σ to Q**

FSA as Transition Table

	Input		
State	b	a	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4:	∅	∅	∅

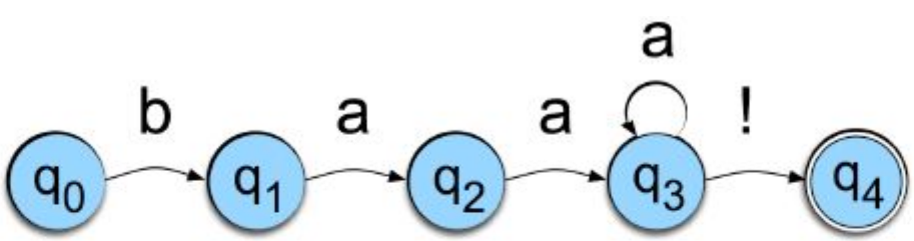
Example, at State 0 :

- **Input 'b' → (transition to) State 1**
- **Input 'a' → No transition (∅) : fail**
- **Input '!' → No transition (∅) :fail**

At State 1:

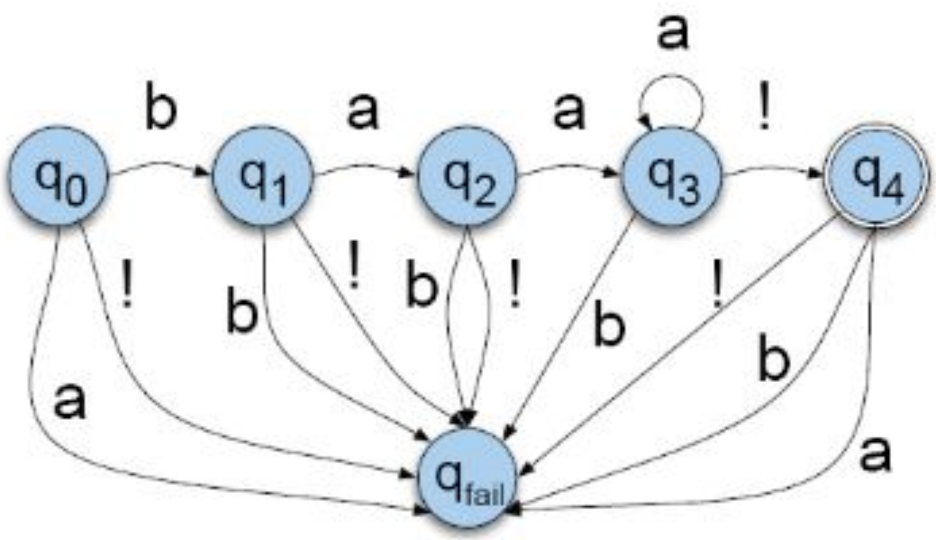
- **Input 'b' → No transition (∅) : fail**
- **Input 'a' → (transition to) State 2**
- **Input '!' → No transition (∅) :fail**

……….



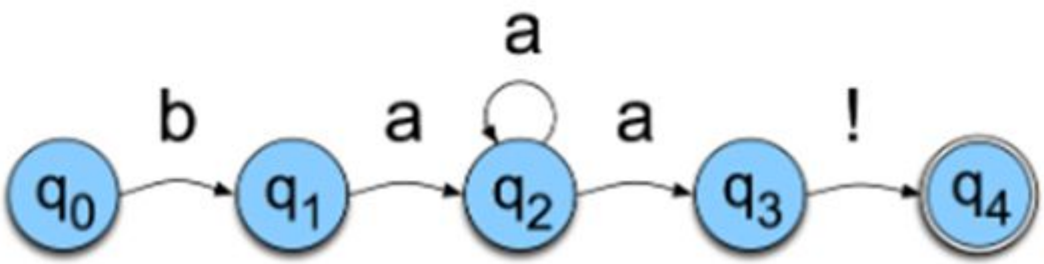
FSA – Fail/Sink State

- States that have no outgoing transitions for certain symbols.
- In such cases we assume that these transitions do exist, but lead to some non-final/non-acceptance state from which you cannot leave.



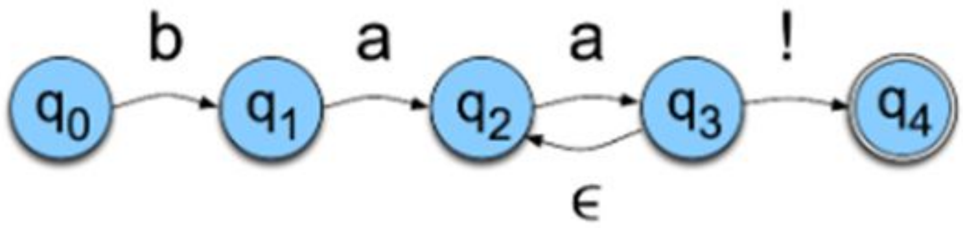
Deterministic FSA

- There is only one possible transition to the next state
- One future state for each state-character pair, and thus there is only one computation path on any given input string.



Non-deterministic FSA

- There is zero, one or more possible transition to the next state
- Produces many different *computation paths* for the same input string; and we say that an NFA accepts a string if *at least one* of those paths ends in an accept state.
- Any DFSA can be turned into a NFSA, so any language accepted by DFSA can be accepted by DFSA

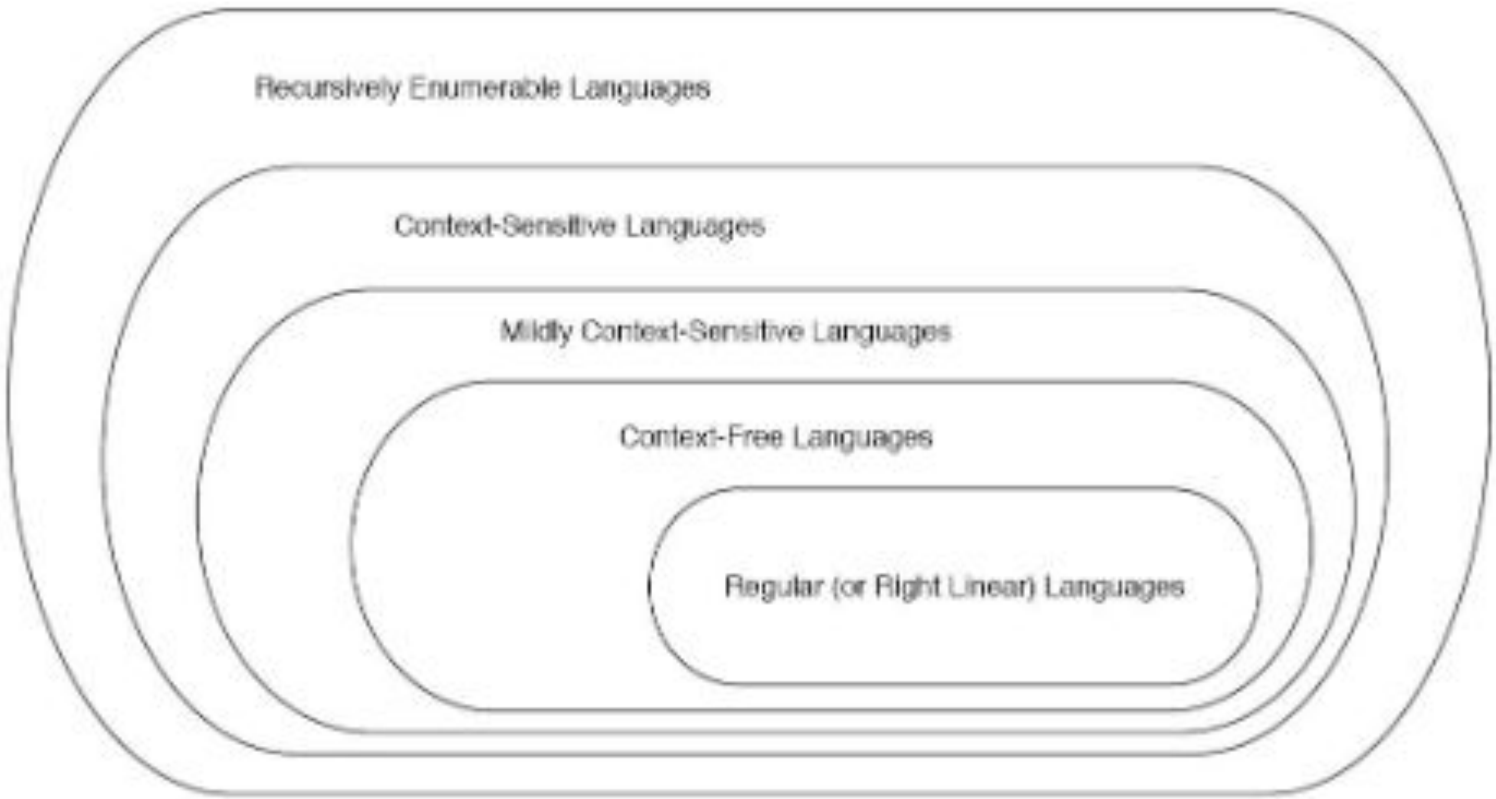


Regular Grammar (Chomsky Hierarchy)

Type	Common Name	Rule Skeleton	Linguistic Example
0	Turing Equivalent	$\alpha \rightarrow \beta$, s.t. $\alpha \neq \epsilon$	HPSG, LFG, Minimalism
1	Context Sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$, s.t. $\gamma \neq \epsilon$	
–	Mildly Context Sensitive		TAG, CCG
2	Context Free	$A \rightarrow \gamma$	Phrase-Structure Grammars
3	Regular	$A \rightarrow xB$ or $A \rightarrow x$	Finite-State Automata

Discourse

Regular Grammars



Usage of FSA in NLP

- Morphology
- Phonology
- Lexical generation
- Automatic Speech Recognition
- POS tagging
- Simplification of CFG
- Information Extraction

Regular Expression and Automata

- Regular expressions can be implemented by the finite-state automaton.
- Finite State Automaton (FSA) a significant tool of computational linguistics.
- Variations: - Finite State Transducers (FST)
- N-gram (Topic 5)
- Hidden Markov Models (Topic 7)

Regular Expressions

- Formal definition:

1. $\emptyset$ is a regular language
2. $\forall a \in \Sigma \cup \epsilon$, $\{a\}$ is a regular language
3. If L_1 and L_2 are regular languages, then so are:
   (a) $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$, the **concatenation** of L_1 and L_2
   (b) $L_1 \cup L_2$, the **union** or **disjunction** of L_1 and L_2
   (c) L_1^*, the **Kleene closure** of L_1

- Language = a set of strings given some alphabet
- If a language is regular, then there must be a regular expression that describes it

Regular Expressions (cont…)

- Regular languages are closed under:

intersection	if L_1 and L_2 are regular languages, then so is $L_1 \cap L_2$, the language consisting of the set of strings that are in both L_1 and L_2.
difference	if L_1 and L_2 are regular languages, then so is $L_1 - L_2$, the language consisting of the set of strings that are in L_1 but not L_2.
complementation	If L_1 is a regular language, then so is $\Sigma^* - L_1$, the set of all possible strings that aren't in L_1.
reversal	If L_1 is a regular language, then so is L_1^R, the language consisting of the set of reversals of all the strings in L_1.

Regular Expressions in NLP

- A compact textual representation of a set of strings that constitute a language
- Used to search for strings that satisfy certain patterns
- Widely used in Computer Science applications:
  - Emacs & vi (e.g., ‘grep’) in UNIX, Perl, Python, etc… (mostly scripting languages)

Regular Expressions in NLP

- Simple but powerful tools for 'shallow processing' (i.e., surface level) of a document or "corpus"
  - What word begins a sentence?
  - What words begin a question?
  - Identify all noun phrases
- Usage:
  - build simple interactive applications (e.g. Eliza chatbot)
  - morphological analysis
  - recognize Named Entities (NE): people names, company names

Exercise 3:

Submit your work individually through Google Classroom by end of class today 19/9/2019 (typed or scanned version)

1. Find the shortest string that is not in the language represented by the regular expression $\mathbf{h^{*}(he)^{*}e^{*}}$.
2. Find a regular expression corresponding to the language of all strings over the alphabet { e, h } that contain exactly two **h**'s **(refer slide #4)**.
3. Find a regular expression corresponding to the language of all strings over the alphabet { e, h} that do not end with **he** **(refer slide #4)**.