Computational Linguistics 1

CMSC/LING 723, LBSC 744

Kristy Hollingshead Seitz
Institute for Advanced Computer Studies
University of Maryland

Lecture 2: 6 September 2011

Agenda

- HW0 – questions? Due Thursday *before class!*
  - When in doubt, keep it simple...
- Regular expressions
- Finite-state automata (deterministic vs. non-deterministic)
- Finite-state transducers
- Set math with FSAs

Agenda

- HW0 – questions? Due Thursday *before class!*
- Regular expressions
- Finite-state automata (deterministic vs. non-deterministic)
- Finite-state transducers
- Set math with FSAs

Regular Expressions

- A meta-language for specifying simple classes of strings
  - Very useful in searching and matching text strings
- Regular expressions are everywhere!
  - Implementations in the shell (sed, awk, bash, grep), Perl, Java, Python, …

Regular Expressions (crash course)

• [a-z]	exactly one lowercase letter
• [a-z]*	zero or more lowercase letters
• [a-z]+	one or more lowercase letters
• [a-z]?	zero or one lowercase letters
• [a-zA-Z0-9]	one lowercase or uppercase letter, or a digit
• [^(]	match anything that is *not* '('

Examples of Regular Expressions

- Basic regular expressions
  /happy/ → happy
  /[abcd]/ → a, b, c, d
  /[a-d]/ → a, b, c, d
  /[^a-d]/ → e, f, g, … z
  /[Tt]he/ → The, the
  /(dog|cat)/ → dog, cat
- Special metacharacters
  /colou?r/ → color, colour
  /oo*h!/ → oh!, ooh!, oooh!, …
  /oo+h!/ → ooh!, oooh!, ooooh!, …
  /beg.n/ → began, begin, begun, begbn, …

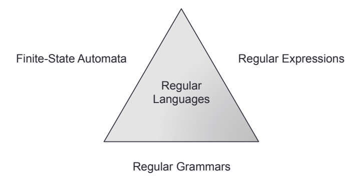
from Jimmy Lin

Agenda

- Regular expressions
- **Finite-state automata** (deterministic vs. non-deterministic)
- Finite-state transducers
- Set math with FSAs

Equivalence Relations

- We can say the following
  - Regular expressions describe a regular language
  - Regular expressions can be implemented by finite-state automata
  - Regular languages can be generated by regular grammars

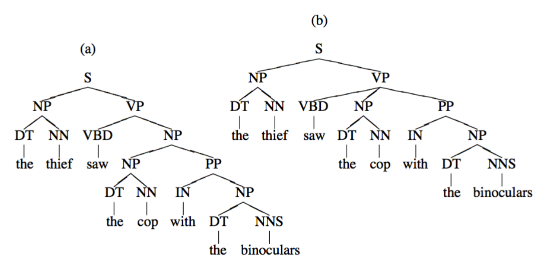


from Jimmy Lin

Chomsky Hierarchy

Language	Mechanisms	Examples
Regular	Regular expressions Regular grammars **Finite-state automata** Finite-state transducers WFSAs/WFSTs	xa^ny Morphology Phonology Taggers
Context-free	Context-free grammars (CFGs) Pushdown automata	a^nb^n Most syntax
Context-sensitive	Unification grammars Lexicalized formalisms (e.g., TAG, CCG)	$a^nb^mc^nd^m$ Cross-serial dependencies

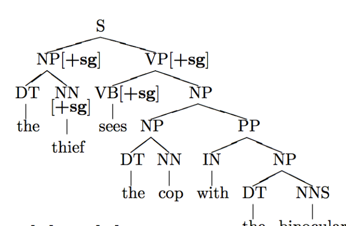
Context-free



Chomsky Hierarchy

Language	Mechanisms	Examples
Regular	Regular expressions Regular grammars **Finite-state automata** Finite-state transducers WFSAs/WFSTs	xa^ny Morphology Phonology Taggers
Context-free	Context-free grammars (CFGs) Pushdown automata	a^nb^n Most syntax
Context-sensitive	Unification grammars Lexicalized formalisms (e.g., TAG, CCG)	$a^nb^mc^nd^m$ Cross-serial dependencies

Context-sensitive: Unification



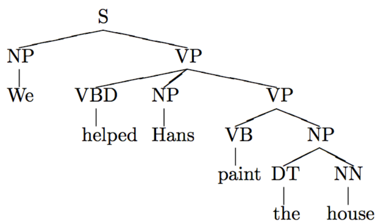
$\mathbf{S}[\alpha] \rightarrow \mathbf{NP}[\alpha]\ \mathbf{VP}[\alpha]$

$\mathbf{NP}[\alpha] \rightarrow \mathbf{DT}\ \mathbf{NN}[\alpha]$

$\mathbf{VP}[\alpha] \rightarrow \mathbf{VB}[\alpha]\ \mathbf{NP}$

the thieves see ...

Context-sensitive: Cross-serial Dependencies

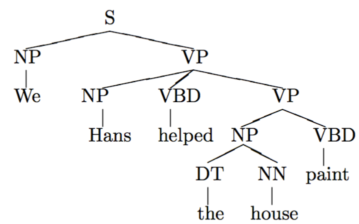


We helped Hans paint the house

We Hans helped the house paint

We Hans the house helped paint

Context-sensitive: Cross-serial Dependencies

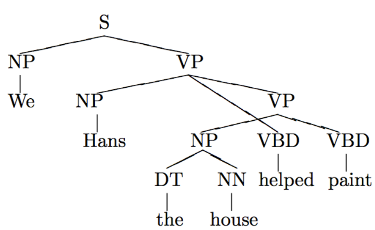


We helped Hans paint the house

We Hans helped the house paint

We Hans the house helped paint

Context-sensitive: Cross-serial Dependencies



We helped Hans paint the house

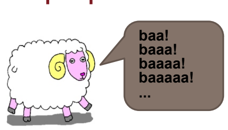
We Hans helped the house paint

We Hans the house helped paint

Chomsky Hierarchy

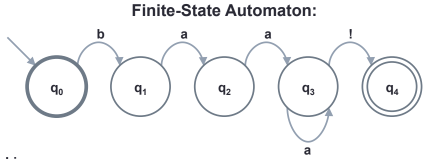
Language	Mechanisms	Examples
Regular	Regular expressions Regular grammars **Finite-state automata** Finite-state transducers WFSAs/WFSTs	xa^ny Morphology Phonology Taggers
Context-free	Context-free grammars (CFGs) Pushdown automata	a^nb^n Most syntax
Context-sensitive	Unification grammars Lexicalized formalisms (e.g., TAG, CCG)	$a^nb^mc^nd^m$ Cross-serial dependencies

Sheep-speech Automaton



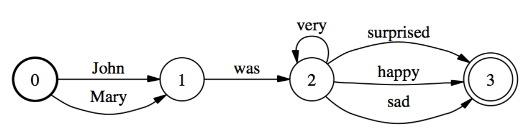
/baa+!/

Finite-State Automaton:

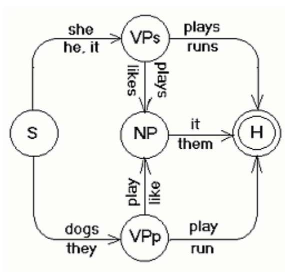


from Jimmy Lin

Natural Language Automaton



Natural Language Automaton

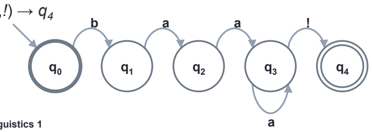


Finite-State Automata (FSA)

- Formal definitions
  - What are they?
  - What do they do?
  - How do they work?

FSA: What are they?

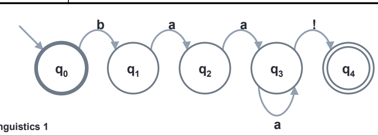
- Q: a finite set of N states
  - Q = {q_0, q_1, q_2, q_3, q_4}
  - The start state: q_0
  - The set of final states: F = {q_4}
- Σ: a finite input alphabet of symbols
  - Σ = {*a*, *b*, *!*}
- δ(*q*,*i*): transition function
  - Given state *q* and input symbol *i*, return new state *q'*
  - $\delta(q_3, !) \rightarrow q_4$



from Jimmy Lin

FSA: State Transition Table

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



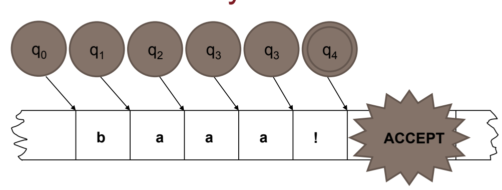
from Jimmy Lin

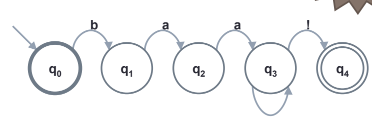
FSA: What do they do?

- Given a string, a FSA either rejects or accepts it
  - ba! → reject
  - baa! → accept
  - baaaz! → reject
  - baaaa! → accept
  - baaaaaa! → accept
  - baa → reject
  - moooo → reject
- Applications in NLP?
  - Grammaticality (on the word level)
  - Morphology (sub-word level)
  - Orthography (character-level)
  - Phonology (phoneme-level)

from Jimmy Lin

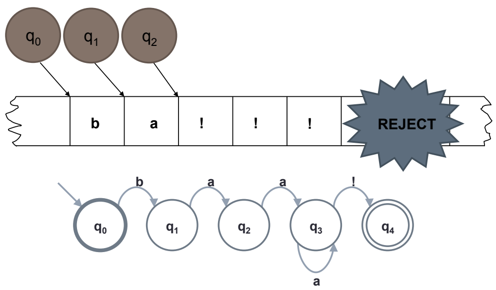
FSA: How do they work?





from Jimmy Lin

FSA: How do they work?



from Jimmy Lin

D-RECOGNIZE

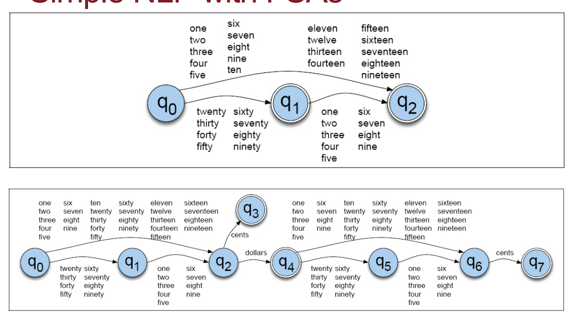
```
function D-RECOGNIZE(tape, machine) returns accept or reject

  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
   if End of input 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
```

from Jimmy Lin

Accept or Generate?

- Formal languages are sets of strings
  - Strings composed of symbols drawn from a finite alphabet
- Finite-state automata define formal languages
  - Without having to enumerate all the strings in the language
- Two views of FSAs:
  - Acceptors to tell you if a string is in the language
  - Generators to produce all and only the strings in the language

Simple NLP with FSAs



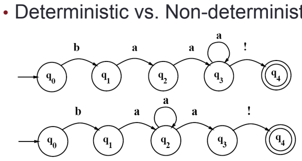
from Jimmy Lin

Agenda

- Regular expressions
- Finite-state automata (deterministic vs. non-deterministic)
- Finite-state transducers
- Set math with FSAs

Introducing Non-Determinism

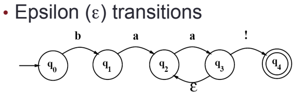
- Deterministic vs. Non-deterministic FSAs



/baa+!/

/ba+a!/

- Epsilon (ε) transitions



Using NFSAs to Accept Strings

- What does it mean?
  - Accept: there exists at least one path (need not be all paths)
  - Reject: no paths exist
- General approaches:
  - Backup: add markers at choice points, then possibly revisit unexplored arcs at marked choice point
  - Look-ahead: look ahead in input to provide clues
  - Parallelism: look at alternatives in parallel
- Recognition with NFSAs as search through state space
  - Agenda holds (state, tape position) pairs

from Jimmy Lin

ND-RECOGNIZE

```
function ND-RECOGNIZE(tape, machine) returns accept or reject

 agenda ← {(Initial state of machine, beginning of tape)}
 current-search-state ← NEXT(agenda)
 loop
   if ACCEPT-STATE?(current-search-state) returns true then
     return accept
   else
     agenda ← agenda ∪ GENERATE-NEW-STATES(current-search-state)
   if agenda is empty then
     return reject
   else
     current-search-state ← NEXT(agenda)
 end
```

from Jimmy Lin

ND-RECOGNIZE

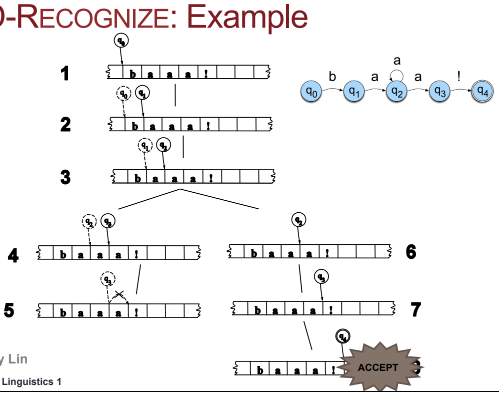
```
function GENERATE-NEW-STATES(current-state) returns a set of search-states

 current-node ← the node the current search-state is in
 index ← the point on the tape the current search-state is looking at
 return a list of search states from transition table as follows:
   (transition-table[current-node,ε], index)
   ∪
   (transition-table[current-node, tape[index]], index + 1)

function ACCEPT-STATE?(search-state) returns true or false

 current-node ← the node search-state is in
 index ← the point on the tape search-state is looking at
 if index is at the end of the tape and current-node is an accept state of machine
then
   return true
 else
   return false
```

from Jimmy Lin

State Orderings

- Stack (LIFO): depth-first
- Queue (FIFO): breadth-first

ND-RECOGNIZE: Example



from Jimmy Lin

What's the point?

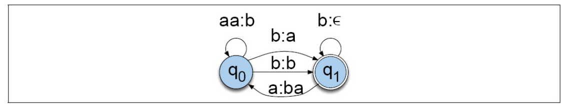
- NFSAs and DFSAs are equivalent
  - For every NFSA, there is a equivalent DFSA (and vice versa)
- Equivalence between regular expressions and FSA
  - Easy to show with NFSAs
- Why use NFSAs?

Agenda

- Regular expressions
- Finite-state automata (deterministic vs. non-deterministic)
- Finite-state transducers
- Set math with FSAs

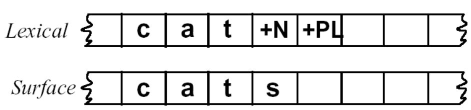
Finite-State Transducers (FSTs)

- A two-tape automaton that recognizes or generates pairs of strings
- Think of an FST as an FSA with two symbol strings on each arc
  - One symbol string from each tape



Four-fold view of FSTs

- As a recognizer
- As a generator
- As a translator
- As a set relater



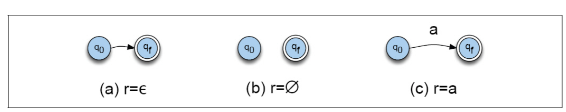
Agenda

- Regular expressions
- Finite-state automata (deterministic vs. non-deterministic)
- Finite-state transducers
- Set math with FSAs

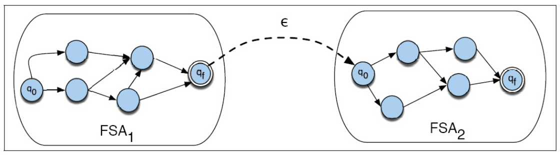
Regular Language: Definition

- Regular languages/FSAs as sets
  - Set math
- $\varnothing$ is a regular language
- $\forall a \in \Sigma \cup \varepsilon$, $\{a\}$ is a regular language
- If L_1 and L_2 are regular languages, then so are:
  - $L_1 \cdot L_2 = \{x\, y \mid x \in L_1 , y \in L_2 \}$, the *concatenation* of L_1 and L_2
  - $L_1 \cup L_2$, the *union* or *disjunction* of L_1 and L_2
  - $L_1{*}$, the *Kleene closure* of L_1

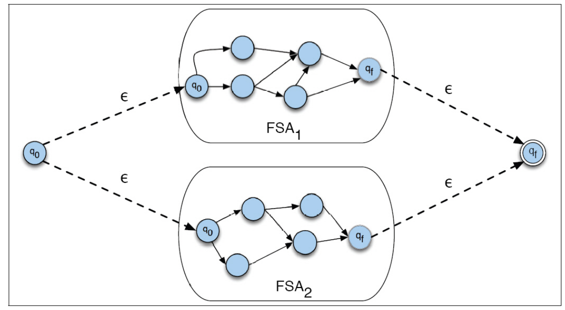
Regular Languages: Starting Points



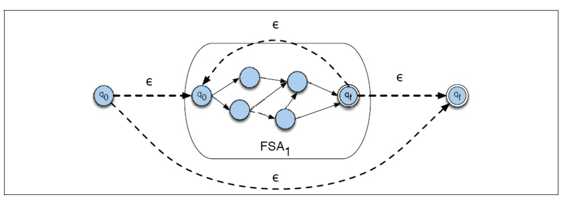
Regular Languages: Concatenation



Regular Languages: Disjunction



Regular Languages: Kleene Closure



Agenda: Summary

- Regular expressions
- Finite-state automata (deterministic vs. non-deterministic)
- Finite-state transducers
- Set math with FSAs