Natural Language Processing

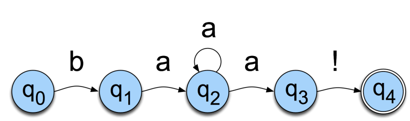
Lecture 3—9/1/2015
Jim Martin

Today

- More FSA material
  - ND-Recognize
  - Determinizing machines
    - Subset construction
  - Composing FSAs
- English morphology
- Morphological processing and FSAs

ND Recognition

- Two basic approaches
  - Take a ND machine and convert it to a D machine and then do recognition with that
  - Explicitly manage the process of recognition as a state-space search (leaving the ND machine as is).



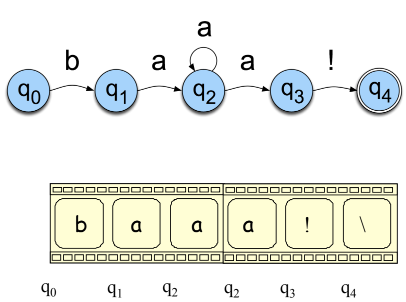
Non-Deterministic Recognition: Search

- In a ND FSA there exists at least one path through the machine for any string that is in the language defined by the machine.
- But not all paths directed through the machine for an accept string necessarily lead to an accept state.
- No paths through the machine lead to an accept state for a string not in the language.

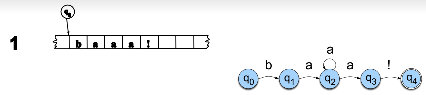
Non-Deterministic Recognition

- So success in non-deterministic recognition occurs when a path is found through the machine that ends in an accept state.
- Failure occurs when all of the possible paths for a given string lead to failure.

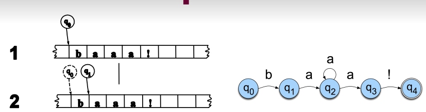
Example



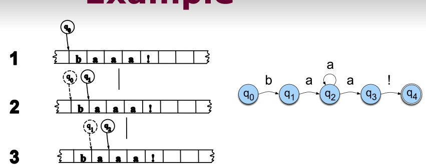
Example



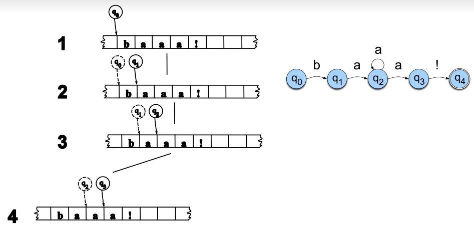
Example



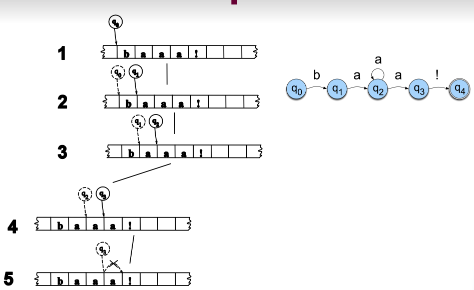
Example



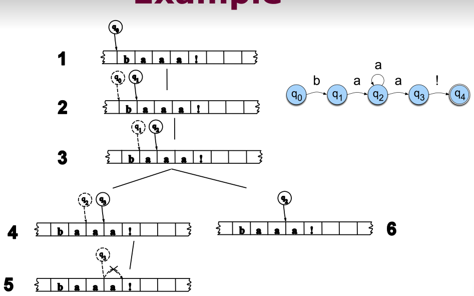
Example



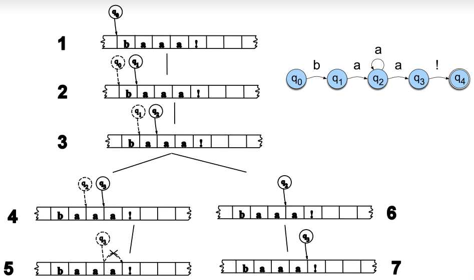
Example



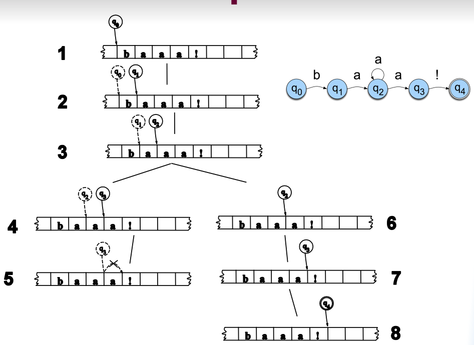
Example



Example



Example

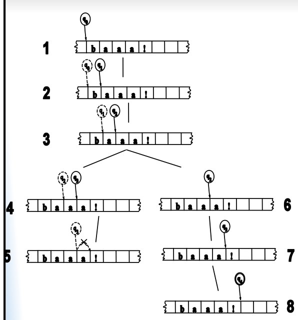


ND-Recognize

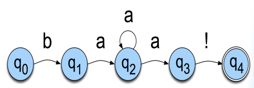
```
function ND-RECOGNIZE(tape, machine) returns accept or reject

 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
```

Example

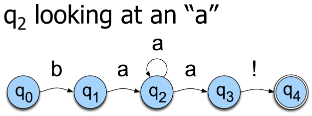


Current	Agenda
	{(q0, 1)}
(q0,1)	{}
	{(q1,2)}
(q1,2)	{}
	{(q2,3)}
(q2,3)	{}
	{(q3,4), (q2,4)}
(q3,4)	{(q2,4)}
(q2,4)	{}
	{(q3,5), (q2,5)}
(q3,5)	{(q2,5)}
	{(q4,6), (q2,5)}
(q4,6)	



Converting NFAs to DFAs

- The Subset Construction is the means by which we can convert an NFA to a DFA automatically.
- The intuition is to think about being in multiple states at the same time. Let's go back to our earlier example where we're in state q_2 looking at an "a"



Subset Construction

- The trick is to simulate going to both q2 and q3 at the same time
- One way to do this is to imagine a new state of a new machine that represents the state of being in states q2 and q3 at the same time
  - Let's call that new state {q2,q3}
    - That's just the name of a new state but it helps us remember where it came from
  - That state is a subset of the original set of states
- The construction does this for all possible subsets of the original states (the powerset).
  - And then we fill in the transition table for that set

Subset Construction

- Given an NFA with the usual parts: Q, Σ, transition function δ, start state q_0, and designated accept states
- We'll construct a new DFA that accepts the same language where
  - States of the new machine are the powerset of states Q: call it Q_D
    - Set of all subsets of Q
  - Start state is $\{q_0\}$
  - Alphabet is the same: Σ
  - Accept states are the states in Q_D that contain *any* accept state from Q

Subset Construction

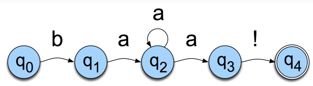
- What about the transition function?
  - For every new state we'll create a transition on a symbol α from the alphabet to a new state as follows
  - $\delta_D(\{q_1,\ldots,q_k\}, \alpha)$ = is the union over all i = 1,…,k of $\delta_N(q_i, \alpha)$ for all α in the alphabet

Baaa!

- How does that work out for our example?
  - Alphabet is still "a", "b" and "!"
  - Start state is {q0}
  - Rest of the states are: {q1}, {q2},... {q4}, {q1,q2}, {q1,q3}... {q0,q1,q2,q3,q4,q5}
    - All 2^5-1 subsets of states in Q
- What's the transition table going to look like?

Lazy Method

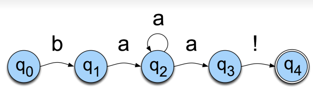
	b	a	!
q0	q1		
q1		q2	
q2		q2,q3	
q3			q4
q4			



	b	a	!
{q0}			

Baaa!

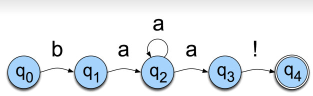
	b	a	!
q0	q1		
q1		q2	
q2		q2,q3	
q3			q4
q4			



	b	a	!
{q0}	{q1}		

Baaa!

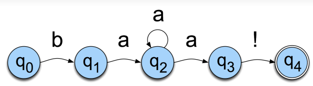
	b	a	!
q0	q1		
q1		q2	
q2		q2,q3	
q3			q4
q4			



	b	a	!
{q0}	{q1}		
{q1}			

Baaa!

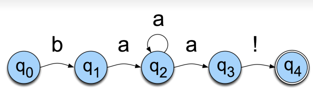
	b	a	!	
q0	q1			
q1		q2		
q2		q2,q3		
q3			q4	
q4				



	b	a	!	
{q0}	{q1}			
{q1}		{q2}		
{q2}				

Baaa!

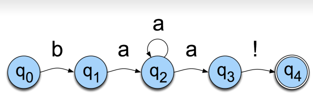
	b	a	!	
q0	q1			
q1		q2		
q2		q2,q3		
q3			q4	
q4				



	b	a	!	
{q0}	{q1}			
{q1}		{q2}		
{q2}		{q2,q3}		
{q2,q3}				

Baaa!

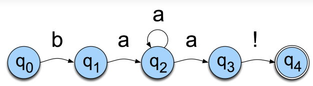
	b	a	!	
q0	q1			
q1		q2		
q2		q2,q3		
q3			q4	
q4				



	b	a	!	
{q0}	{q1}			
{q1}		{q2}		
{q2}		{q2,q3}		
{q2,q3}		{q2,q3}		

Baaa!

	b	a	!
q0	q1		
q1		q2	
q2		q2,q3	
q3			q4
q4			



	b	a	!
{q0}	{q1}		
{q1}		{q2}	
{q2}		{q2,q3}	
{q2,q3}		{q2,q3}	{q4}
{q4}			

Couple of Notes

- We didn't come close to needing 2^Q new states. Most of those were unreachable. So in theory there is the potential for an explosion in the number of states. In practice, it may be more manageable.
- Draw the new deterministic machine from the table on the previous slide... It should look familiar.

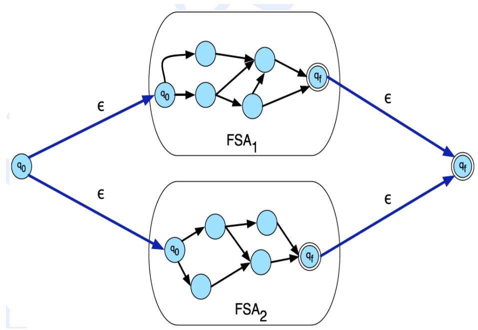
Compositional Machines

- Recall that formal languages are just sets of strings
- Therefore, we can talk about set operations (intersection, union, concatenation, negation) on languages
- This turns out to be a very useful
  - It allows us to decompose problems into smaller problems, solve those problems with specific languages, and then compose those solutions to solve the big problems.

Example

- Create a regex to match all the ways that people write down phone numbers. For just the U.S. that needs to cover
  - (303) 492-5555
  - 303.492.5555
  - 303-492-5555
  - 1-303-492-5555
  - (01) 303-492-5555
- You could write a big hairy regex to capture all that, or you could write individual regex's for each type and then OR them together into a new regex/machine.
- How does that work?

Union (Or)



Negation

- Construct a machine M2 to accept all strings not accepted by machine M1 and reject all the strings accepted by M1
  - Invert all the accept and not accept states in M1
- Does that work for non-deterministic machines?

Intersection (AND)

- Accept a string that is in both of two specified languages
- An indirect construction…
  - A^B = ~(~A or ~B)

Changing Gears

- Let's switch to talking about why this stuff is relevant to NLP
- In particular, we'll be talking about words and morphology

Words

- Finite-state methods are particularly useful in dealing with large lexicons
  - That is, big *bunches of words*
  - Often infinite sized bunches
- Many devices, some with limited memory resources, need access to large lists of words
- And they need to perform fairly sophisticated tasks with those lists
- Let's first talk about some facts about words and then come back to computational methods

English Morphology

- Morphology is the study of the ways that words are built up from smaller units called morphemes
  - The *minimal meaning-bearing* units in a language
- We can usefully divide morphemes into two classes
  - Stems: The core meaning-bearing units
  - Affixes: Bits and pieces that adhere to stems to change their meanings and grammatical functions

English Morphology

- We can further divide morphology up into two broad classes
  - Inflectional
  - Derivational

Word Classes

- By word class, we have in mind familiar notions like noun and verb
  - Also referred to as parts of speech and lexical categories
- We'll go into the gory details in Chapter 5
- Right now we're concerned with word classes because the way that stems and affixes combine is based to a large degree on the word class of the stem

Inflectional Morphology

- Inflectional morphology concerns the combination of stems and affixes where the resulting word....
  - Has *the same word class* as the original
  - And serves a grammatical/semantic purpose that is
    - *Different* from the original
    - But is nevertheless *transparently* related to the original
      - "walk" + "s" = "walks"

Inflection in English

- Nouns are simple
  - Markers for plural and possessive
- Verbs are only slightly more complex
  - Markers appropriate to the tense of the verb
- That's pretty much it
  - Other languages can be quite a bit more complex
  - An implication of this is that hacks (approaches) that work in English will not work for many other languages

Regulars and Irregulars

- Things are complicated by the fact that some words misbehave (refuse to follow the rules)
  - Mouse/mice, goose/geese, ox/oxen
  - Go/went, fly/flew, catch/caught
- The terms *regular* and *irregular* are used to refer to words that follow the rules and those that don't

Regular and Irregular Verbs

- Regulars…
  - Walk, walks, walking, walked, walked
- Irregulars
  - Eat, eats, eating, ate, eaten
  - Catch, catches, catching, caught, caught
  - Cut, cuts, cutting, cut, cut

Inflectional Morphology

- So inflectional morphology in English is fairly straightforward
- But is complicated by the fact that are irregularities

Derivational Morphology

- Derivational morphology is the messy stuff that no one ever taught you
- In English it is characterized by
  - Quasi-systematicity
  - Irregular meaning change
  - Changes of word class

Derivational Examples

- Verbs and Adjectives to Nouns

-ation	computerize	computerization
-ee	appoint	appointee
-er	kill	killer
-ness	fuzzy	fuzziness

Derivational Examples

- Nouns and Verbs to Adjectives

-al	computation	computational
-able	embrace	embraceable
-less	clue	clueless

Example: *Compute*

- Many paths are possible…
- Start with compute
  - Computer -> computerize -> computerization
  - Computer -> computerize -> computerizable
- But not all paths/operations are equally good (allowable?)
  - Clue
    - Clue → clueless
    - Clue → ?clueful
    - Clue → *clueable

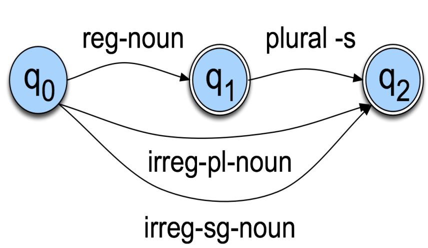
Morphology and FSAs

- We would like to use the machinery provided by FSAs to capture these facts about morphology
  - Accept strings that are in the language
  - Reject strings that are not
  - And do so in a way that doesn't require us to list all the forms of all the words in the language
    - Even in English this is inefficient
    - And in other languages it is impossible

Start Simple

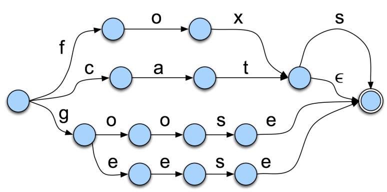
- Regular singular nouns are ok as is
  - They are in the language
- Regular plural nouns have an -s on the end
  - So they're also in the language
- Irregulars are ok as is

Simple Rules



Now Plug in the Words Spelled Out

Replace the class names like “reg-noun” with FSAs that recognize all the words in that class.



9/1/15 Speech and Language Processing - Jurafsky and Martin 52