Lecture 4: Finite State Technology

CSA3202 Human Language Technology

Mike Rosner, Dept ICS
October 2011
Acknowledgements

- Jurafsky and Martin Chapters 2 and 3 (diagrams)
Outline

1 Finite State Automata

2 Finite State Morphological Parsing
Exercise 3.1

Write regular expressions, including definition of alphabet, for

1. The language \{walk, walks, walking, walked\}
   \[\Sigma = \{\epsilon, g, l, k, n, s, w, a, e, i\}\]
   \[L_1 = (((w.a).l).k). (\epsilon + s + ((i.n).g) + (e.d))\]

2. The same language for the verb "sleep": {sleep, sleeps, sleeping, slept}
   \[\Sigma = \{\epsilon, g, l, n, p, s, e, i\}\]
   \[L_2 = (((((s.l).e).e).p). (\epsilon + s + ((i.n).g) + (e.d))\]
   \[\Sigma = \{\epsilon, g, l, n, p, s, t, e, i\}\]

3. The same language for the verb "run": {run, runs, running, ran}
   \[\Sigma = \{\epsilon, g, n, r, s, t, a, e, i, u\}\]
   \[L_3 = ((r.u).n). (\epsilon + s + (((n.i).n).g)) + ((r.a).n)\]

4. The language \{nikteb, tikteb, jikteb, niktbu, tiktbu, jiktbu\}
Write regular expressions, including definition of alphabet, for

1. The language \{walk, walks, walking, walked\}
   \[ \Sigma = \{ \epsilon, g, l, k, n, s, w, a, e, i \} \]
   \[ L_1 = (((w.a).l).k). (\epsilon + s + ((i.n).g) + (e.d)) \]

2. The same language for the verb “sleep”:
   \{sleep, sleeps, sleeping, slept\}
   \[ \Sigma = \{ \epsilon, g, l, n, p, s, e, i \} \]
   \[ L_2 = ((((s.l).e).e).p). (\epsilon + s + ((i.n).g) + (e.d)) \]
   \[ \Sigma = \{ \epsilon, g, l, n, p, s, t, e, i \} \]
   \[ L_2 = ((((s.l).e).e).p). (\epsilon + s + ((i.n).g) + (e.d)) + ((((s.l).e).p).t) \]

3. The same language for the verb “run”:
   \{run, runs, running, ran\}
   \[ \Sigma = \{ \epsilon, g, n, r, s, t, a, e, i, u \} \]
   \[ L_3 = ((r.u).n). (\epsilon + s + (((n.i).n).g)) + ((r.a).n) \]

4. The language \{nikteb, tikteb, jikteb, niktbu, tiktbu, jiktbu\}
Exercise 3.1

Write regular expressions, including definition of alphabet, for

1. The language \{walk, walks, walking, walked\}
\[ \Sigma = \{ \epsilon, g, l, k, n, s, w, a, e, i \} \]
\[ L_1 = (((w.a).l).k).\epsilon + s + ((i.n).g) + (e.d) \]

2. The same language for the verb "sleep":
\{sleep, sleeps, sleeping, slept\}
\[ \Sigma = \{ \epsilon, g, l, n, p, s, e, i \} \]
\[ L_2 =((((s.l).e).e).p).\epsilon + s + ((i.n).g) + (e.d) \]
\[ \Sigma = \{ \epsilon, g, l, n, p, s, t, e, i \} \]

3. The same language for the verb "run":
\{run, runs, running, ran\}
\[ \Sigma = \{ \epsilon, g, n, r, s, t, a, e, i, u \} \]
\[ L_3 = ((r.u).n).\epsilon + s + (((n.i).n).g)) + ((r.a).n) \]

4. The language \{nikteb, tikteb, jikteb, niktbu, tiktbu, jiktbu\}
Exercise 3.1

Write regular expressions, including definition of alphabet, for

1. The language \{walk, walks, walking, walked\}
   \[\Sigma = \{\epsilon, g, l, k, n, s, w, a, e, i\}\]
   \[L_1 = (((w.a).l).k).((\epsilon + s + ((i.n).g) + (e.d))\]

2. The same language for the verb “sleep":
   \{sleep, sleeps, sleeping, slept \}
   \[\Sigma = \{\epsilon, g, l, n, p, s, e, i\}\]
   \[L_2 = (((s.l).e).e).p).((\epsilon + s + ((i.n).g) + (e.d))\]
   \[\Sigma = \{\epsilon, g, l, n, p, s, t, e, i\}\]

3. The same language for the verb “run":
   \{run, runs, running, ran \}
   \[\Sigma = \{\epsilon, g, n, r, s, t, a, e, i, u\}\]
   \[L_3 = ((r.u).n).((\epsilon + s + ((n.i).n).g)) + ((r.a).n)\]

4. The language \{nikteb, tikteb, jikteb, niktbu, tiktbu, jiktbu\}
Exercise 3.1

Write regular expressions, including definition of alphabet, for

1. The language \{walk, walks, walking, walked\}
   \[ \Sigma = \{ \epsilon, g, l, k, n, s, w, a, e, i \} \]
   \[ L_1 = (((w.a).l).k).(\epsilon + s + ((i.n).g) + (e.d)) \]

2. The same language for the verb “sleep”:
   \{sleep, sleeps, sleeping, slept\}
   \[ \Sigma = \{ \epsilon, g, l, n, p, s, e, i \} \]
   \[ L_2 = (((s.l).e).e).p).(\epsilon + s + ((i.n).g) + (e.d)) \]
   \[ \Sigma = \{ \epsilon, g, l, n, p, s, t, e, i \} \]
   \[ L_2 = (((s.l).e).e).p).(\epsilon + s + ((i.n).g)) + (((s.l).e).p).t \]

3. The same language for the verb “run”:
   \{run, runs, running, ran\}
   \[ \Sigma = \{ \epsilon, g, n, r, s, t, a, e, i, u \} \]
   \[ L_3 = ((r.u).n).(\epsilon + s + (((n.i).n).g)) + ((r.a).n) \]

4. The language \{nikteb, tikteb, jikteb, niktbu, tiktbu, jiktbu\}
Exercise 3.1

Write regular expressions, including definition of alphabet, for

1. The language \{walk, walks, walking, walked\}
   \[ \Sigma = \{ \epsilon, g, l, k, n, s, w, a, e, i \} \]
   \[ L_1 = (((w.a).l).k). (\epsilon + s + ((i.n).g) + (e.d)) \]

2. The same language for the verb “sleep”:
   \{sleep, sleeps, sleeping, slept\}
   \[ \Sigma = \{ \epsilon, g, l, n, p, s, e, i \} \]
   \[ L_2 = (((s.l).e).e).p). (\epsilon + s + ((i.n).g) + (e.d)) \]
   \[ \Sigma = \{ \epsilon, g, l, n, p, s, t, e, i \} \]

3. The same language for the verb “run”:
   \{run, runs, running, ran\}
   \[ \Sigma = \{ \epsilon, g, n, r, s, t, a, e, i, u \} \]
   \[ L_3 = ((r.u).n). (\epsilon + s + (((n.i).n).g)) + ((r.a).n) \]

4. The language \{nikteb, tikteb, jikteb, niktbu, tiktbu, jiktbu\}
Remarks

- Does $\Sigma$ have to be different for each language?
- Irregularities have to be dealt with separately
- ... but sometimes irregularities are regular (run, stun, gun etc)
- expressions can be difficult to read - notation matters
Automata are models of computation.

A finite state automaton (FSA) is a five-tuple<br>\(< Q, q_0, \Sigma, \delta, F >\), where

- \( Q \) is a set of states
- \( q_0 \in Q \) is an initial state
- \( F \subseteq Q \) is a set of final states
- \( \Sigma \) is a finite set of symbols
- \( \delta \) is a relation \( Q \times \Sigma \times Q \)
Finite State Automata
Finite State Morphological Parsing

FSA Example

Example

\[ Q = \{1, 2, 3\} \]
\[ q_0 = 3 \]
\[ F = \{1\} \]
\[ \Sigma = \{a, b\} \]
\[ \delta = \{(3, a, 2), (2, b, 3), (2, a, 1)\} \]
Language Accepted by an FSA

- Define the reflexive transitive extension $\Delta$ of $\delta$
  - for every state $q \in Q$, $(q, \epsilon, q) \in \Delta$
  - for every string $w \in \Sigma^*$ and letter $a \in \Sigma$, if $(q, w, q') \in \Delta$ and $(q', a, q'') \in \delta$ then $(q, w.a, q'') \in \Delta$
- A string $w$ is accepted by an automaton if and only if there exists $q_f \in Q$ such that
  $$(q_0, w, q_f) \in \Delta$$
- The language accepted by a finite-state automaton is the set of all strings it accepts
- Theorem (Kleene, 1956): The class of languages recognized by finite-state automata is the class of regular languages.
The proof that regular expressions are equivalent to FSAs

1. Show that an automaton can be built for each regular language
2. Show that for each regular language a regular expression can be built

To do the first part we need a base case and then a series of cases for each regular expression operator

---

1 see Hopcraft and Ullman 1979
Base Case

Here we see the how FSAs that are equivalent to three cases of regular expressions without operators.

(a) $r=\epsilon$

(b) $r=\emptyset$

(c) $r=a$
Concatenation ($r_1.r_2$)

This is the case of $r_1.r_2$ where $r_1$ is equivalent to FSA1 and $r_2$ to FSA2.
Iteration (Kleene *)

Finite State Automata
Finite State Morphological Parsing

CSA3202 Human Language Technology
Alternation \((r_1 + r_2)\)
### Morphological Analysis Examples

<table>
<thead>
<tr>
<th>English</th>
<th>Morphological Parse</th>
<th>Spanish</th>
<th>Morphological Parse</th>
<th>Gloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>cats</td>
<td>cat +N +PL</td>
<td>pavo</td>
<td>pavo +N +Masc +Pl</td>
<td>‘ducks’</td>
</tr>
<tr>
<td>cat</td>
<td>cat +N +SG</td>
<td>pavo</td>
<td>pavo +N +Masc +Sg</td>
<td>‘duck’</td>
</tr>
<tr>
<td>cities</td>
<td>city +N +Pl</td>
<td>bebo</td>
<td>beber +V +PInd +1P +Sg</td>
<td>‘I drink’</td>
</tr>
<tr>
<td>geese</td>
<td>goose +N +Pl</td>
<td>canto</td>
<td>cantar +V +PInd +1P +Sg</td>
<td>‘I sing’</td>
</tr>
<tr>
<td>goose</td>
<td>goose +N +Sg</td>
<td>canto</td>
<td>canto +N +Masc +Sg</td>
<td>‘song’</td>
</tr>
<tr>
<td>goose</td>
<td>goose +V</td>
<td>puse</td>
<td>poner +V +Perf +1P +Sg</td>
<td>‘I was able’</td>
</tr>
<tr>
<td>gooses</td>
<td>goose +V +3P +Sg</td>
<td>vino</td>
<td>venir +V +Perf +3P +Sg</td>
<td>‘he/she came’</td>
</tr>
<tr>
<td>merging</td>
<td>merge +V +PresPart</td>
<td>vino</td>
<td>vino +N +Masc +Sg</td>
<td>‘wine’</td>
</tr>
<tr>
<td>caught</td>
<td>catch +V +PastPart</td>
<td>lugar</td>
<td>lugar +N +Masc +Sg</td>
<td>‘place’</td>
</tr>
<tr>
<td>caught</td>
<td>catch +V +Past</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. **Lexicon**: list of stems and affixes. This is essentially a list of free and bound morphemes together with information about them.

2. **Morphotactics**: model of morpheme ordering. E.g. that in English the plural morpheme follows the noun rather than precedes it. Morphotactics can be modelled using an FSA.

3. **Orthographic Rules**: These are “spelling" rules that govern changes that occur at morpheme boundaries e.g. die + ing = dying.
In English, noun inflection concerns plurals

- Nouns are either
  - **regular**: to form plural - add s;
  - **irregular**: either (i) special plural form, or (ii) no change
FSA for English Noun Inflection

<table>
<thead>
<tr>
<th>reg-noun</th>
<th>irreg-pl-noun</th>
<th>irreg-sg-noun</th>
<th>plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>fox</td>
<td>geese</td>
<td>goose</td>
<td>-s</td>
</tr>
<tr>
<td>cat</td>
<td>sheep</td>
<td>sheep</td>
<td></td>
</tr>
<tr>
<td>aardvark</td>
<td>mice</td>
<td>mouse</td>
<td></td>
</tr>
</tbody>
</table>

Finite State Automata
Finite State Morphological Parsing

CSA3202 Human Language Technology

4 Finite State Technology

17/27
For verbs, the lexicon has

- **three stem classes**
  1. reg-verb-stem
  2. irreg-verb-stem
  3. irreg-verb-form

- **four affix classes**
  1. -ed past
  2. -ed participle
  3. -ing participle
  4. -s third-singular
FSA for English Verb Inflection

Inflectional

<table>
<thead>
<tr>
<th>reg-verb-stem</th>
<th>irreg-verb-stem</th>
<th>irreg-past-stem</th>
<th>past</th>
<th>past-part</th>
<th>pres-part</th>
<th>3sg</th>
</tr>
</thead>
<tbody>
<tr>
<td>walk</td>
<td>cut</td>
<td>caught</td>
<td>-ed</td>
<td>-ed</td>
<td>-ing</td>
<td>-s</td>
</tr>
<tr>
<td>fry</td>
<td>speak</td>
<td>ate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>talk</td>
<td>sing</td>
<td>eaten</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>impeach</td>
<td></td>
<td>sang</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In general, derivational is more complex and more problematic (for English)

Adjectives represent a simplest cases
### Adjectives

#### Derivational

<table>
<thead>
<tr>
<th>unmarked</th>
<th>comparative</th>
<th>superlative</th>
<th>adverb</th>
</tr>
</thead>
<tbody>
<tr>
<td>happy</td>
<td>happier</td>
<td>happiest</td>
<td>happily</td>
</tr>
<tr>
<td>unhappy</td>
<td>unhappier</td>
<td>unhappiest</td>
<td>unhappily</td>
</tr>
<tr>
<td>clear</td>
<td>clearer</td>
<td>clearest</td>
<td>clearly</td>
</tr>
<tr>
<td>unclear</td>
<td>unclearer</td>
<td>unclearest</td>
<td>unclearly</td>
</tr>
<tr>
<td>cool</td>
<td>cooler</td>
<td>coolest</td>
<td>cooly</td>
</tr>
<tr>
<td>uncool</td>
<td>uncooler</td>
<td>uncoolest</td>
<td>uncoolly</td>
</tr>
</tbody>
</table>

- There is a certain amount of regularity
- Hypothesis: prefix . stem . \((\epsilon + er + est + ly)\)
FSA for Adjectives
Derivational
### Problems

#### Derivational

<table>
<thead>
<tr>
<th>unmarked</th>
<th>comparative</th>
<th>superlative</th>
<th>adverb</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>redder</td>
<td>reddest</td>
<td>redly</td>
</tr>
<tr>
<td>unred</td>
<td>unredder</td>
<td>unreddest</td>
<td>unredly</td>
</tr>
<tr>
<td>big</td>
<td>bigger</td>
<td>biggest</td>
<td>bigly</td>
</tr>
<tr>
<td>unbigr</td>
<td>unbigger</td>
<td>unbiggest</td>
<td>unbigly</td>
</tr>
</tbody>
</table>
Problems

- **FSA overgenerates**: it will recognise forms like *unbig, smally*
  - **Solution**: set up classes of roots and specify their possible suffixes such as
    - adj-root1: adjectives that can occur with un- and -ly
    - adj-root2: adjectives that cannot so occur

- **FSA misses generalisations** such as:
  - adjectives and some nouns (capital) ending in -al or -able can take suffix -ity (equal, formal)
  - but sometimes -ness (naturalness)
  - verbs ending in -ize can be followed by -ation (realize, realization)

  - **Solution**: Introduce extra paths and subclasses to handle these cases
Morphotactict FSA for Fragment of English Derivational Morphology
Handling the Words

- We can use these FSAs to solve the problem of morphological recognition.
- We do this by plugging *sub-lexicons* into the morphotactic FSAs defined earlier.
- Given the right infrastructure, this kind of operation can be performed both *algebraically* and *computationally*. 
Summary

- FSAs can be used to represent both lexicon and morphotactics
- Advantages of FSAs
  - Reversibility: they can be used both for analysis and for generation.
  - Efficiency: most algorithms on finite-state automata (e.g. recognition) are linear.
  - Most phonological and morphological process of natural languages can be straightforwardly described using the operations under which regular languages are closed.
  - The closure properties of regular languages naturally support modular development of finite-state grammars.