CMPT-413: Computational Linguistics

Anoop Sarkar

anoop@cs.sfu.ca

www.sfu.ca/˜anoop/courses/CMPT-413-Spring-2003.html

Formal languages and Computational Linguistics

Formal Language theory	CL
Language	Data/corpus (finite)
Grammar	Grammar (inferred from data, produces infinite set of strings)
Automata	Recognition/Generation Algorithms

Some key definitions

- **Tagging/Classification**: Assigning to the input one out of a finite number of classes
  e.g. *formalization* $\rightarrow$ Noun

- **Parsing**: Assigning a complex structure to some input. Very similar to tagging except the class in this case is itself decomposed into sub-parts.
  e.g. *formalization* $\rightarrow$ formal +Adj +ize +V +ation +N

- **Grammar development**: taking some linguistic data and producing a grammar

Grammar Development: Inflectional morphology

- Write an NFA for the following data such that each suffix type gets a single transition (e.g. adding an *-s* is the plural suffix):

  ```
  cat cats
  dog dogs
  fox foxes
  mouse mice
  ```

- Note that `foxes` is not an isolated case (irregular), e.g. `suffix`, `suffixes`

Grammar Development: Derivational morphology

- Write an NFA for the following data (ignore the parts of speech, also you can use substrings on the transitions, e.g. a transition can have `demon` on it):

```
demon/N demon+ize/V
demon+ize+ation/N demon+ize+able/A
demon+ize+er/N
formal/A formal+ity/N
formal+ness/N
```

Grammar Development: Derivational morphology

- Does your NFA accept the following additional strings (ignore the parts of speech). If not, what do you need to add to your previous NFA?

```
formal+ize/V formal+ize+ation/N
formal+ize+able/A formal+ize+er/N
demon+ize+able+ity/N
```

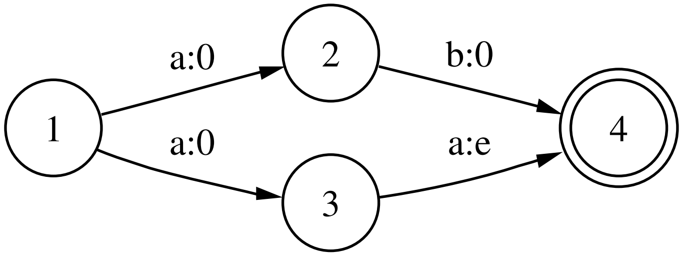
Dealing with `foxes`: Finite-state transducers

- `a:0` is notation for a map between two alphabets: Σ_1 and Σ_2

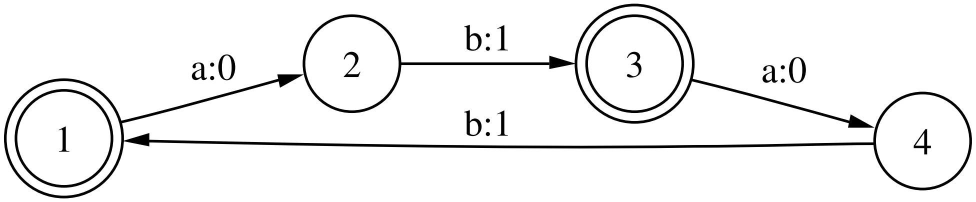
- FSTs accept pairs of strings. Language accepted by an FST:
  $L \subseteq (\Sigma_1^*, \Sigma_2^*)$
  can also be written as a mapping relation between an *input* language and and *output* language: e.g. $\Sigma_1^* \rightarrow \Sigma_2^*$

Dealing with `foxes`: Finite-state transducers

- FST for $(ab, \texttt{00}), (aa, \texttt{0})$



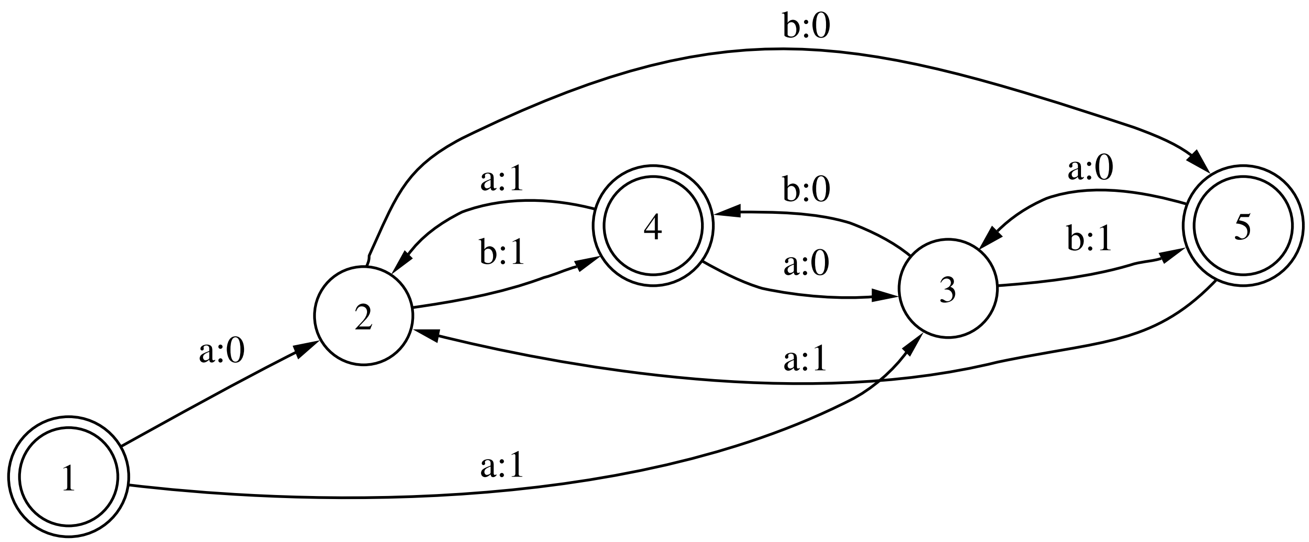
- FST for $(\epsilon, \epsilon), (ab, \texttt{01}), (abab, \texttt{0101}), \ldots$ $\texttt{(ab)}^* \to \texttt{(01)}^*$



Dealing with `foxes`: Finite-state transducers

- Draw FST for $(\epsilon,\epsilon),(ab,\texttt{00}),(ab,\texttt{01}),(ab,\texttt{10}),(ab,\texttt{11}),(abab,\texttt{0000}),\ldots$

  `(ab)`$^* \rightarrow$ `([01][01])*`



Finite-state Transducers

- The mystery transducer: what does it do?

Morphological Parsing with Transducers

- Simpler to start by thinking of it as generation
- Start with `cat +N +PL` and then use a FST to produce `cats`
- Advantage: since we can add/delete material, we can handle `fox +N +PL` to get the correct form `foxes`
- To deal with orthographic rules (like the one above), J&M provide an analysis which is made very modular with the use of an intermediate FST

Morphological Parsing with Transducers

- Draw a transducer for the following examples:

Input	Output	Input	Output
`cat+N+PL`	cat^s#	`cat+N+SG`	cat#
`dog+N+PL`	dog^s#	`dog+N+SG`	dog#
`fox+N+PL`	fox^s#	`fox+N+SG`	fox#
`mouse+N+PL`	mice#	`mouse+N+SG`	mouse#

Morphological Parsing with Transducers

- A transducer for the e-insertion rule
  if word ends in `x^s#` then output `xes`; similarly for `z^#` and `s^#`
  note the use of the intermediate output from the previous transducer
  define `other` = `[a-r,t-w,y]`

Input	Inter.	Output	Input	Inter.	Output
`cat+N+PL`	cat^s#	*cats*	`cat+N+SG`	cat#	*cat*
`fox+N+PL`	fox^s#	*foxes*	`fox+N+SG`	fox#	*fox*

Ambiguity when Parsing with FSTs

- Global ambiguity:
  foxes $\longrightarrow$ `fox+N+PL` OR `foxes+V+3SG`
  I saw two foxes yesterday
  That trickster foxes me every time

- Local ambiguity:
  assess has a prefix string which can be analyzed:
  `ass+N+PL` $\longrightarrow$ *asses*

- An FST will return the two answers in the first case, but only return one answer in the second case (even though it will consider a false analysis partway through the string)

Deterministic vs. Non-deterministic

- Deterministic transducers are called **subsequential** transducers (no backtracking when translating one string to another)

- Deterministic transducers where all the states are final states are called **sequential** transducers.

Porter Stemmer

- Unlike our previous FSTs, the Porter Stemmer has no stems
  This makes the FST much smaller as a result – leading to a simple implementation (available widely on the web in many programming languages)

- *ational* $\longrightarrow$ *ate*
  ing $\longrightarrow \epsilon$ if stem contains vowel (e.g. motoring, motor)

- Performs well enough most of the time, but suffers from problems that the FSTs we saw earlier do not have: *organization* $\longrightarrow$ *organ*
  Still, it is used often for quick and dirty stemming in many NLP applications due to its simplicity and speed

FST Software in Research and Industry

- FSTs are used for many applications in NLP: morphology, stemming, segmentation

- FST software:

```
Van Noord fsa
    URL:http://odur.let.rug.nl/˜Evannoord/Fsa/
AT&T fsm toolkit
    URL:http://www.research.att.com/sw/tools/fsm/
Xerox LinguistX
    URL:http://www.inxight.com/products/oem/linguistx/
Teragram
    URL:http://www.teragram.com/
```

- FSTs are also widely used in aligning sequences in genomics