Finite-State Methods
Finite state acceptors (FSAs)

- Things you may know about FSAs:
  - Equivalence to regexps
  - Union, Kleene *, concat, intersect, complement, reversal
  - Determinization, minimization
  - Pumping, Myhill-Nerode

Diagram:

```
   a
   ^
   |  ε
   v
   c
```

Defines the language \( a? c^* \)

\[ \{a, ac, acc, accc, \ldots, \varepsilon, c, cc, ccc, \ldots\} \]
n-gram models not good enough

- Want to model grammaticality
- A “training” sentence known to be grammatical:
  
  BOS mouse traps catch mouse traps EOS

  trigram model must allow these trigrams

- Resulting trigram model has to overgeneralize:
  - allows sentences with 0 verbs
    BOS mouse traps EOS
  - allows sentences with 2 or more verbs
    BOS mouse traps catch mouse traps
    catch mouse traps catch mouse traps EOS
  - Can’t remember whether it’s in subject or object
    (i.e., whether it’s gotten to the verb yet)
Finite-state models can “get it”

- Want to model grammaticality
  
  \[
  \text{BOS } \text{mouse traps catch mouse traps } \text{EOS}
  \]

- Finite-state can capture the generalization here:

  \[
  \text{Noun}^+ \text{ Verb Noun}^+
  \]

  ![Diagram](image)

  - preverbal states (still need a verb to reach final state)
  - postverbal states (verbs no longer allowed)

  Allows arbitrarily long NPs (just keep looping around for another Noun modifier).

  Still, never forgets whether it’s preverbal or postverbal!

  (Unlike 50-gram model)
How powerful are regexps / FSAs?

- More powerful than n-gram models
  - The hidden state may “remember” arbitrary past context
  - With k states, can remember which of k “types” of context it’s in

- Equivalent to HMMs
  - In both cases, you observe a sequence and it is “explained” by a hidden path of states. The FSA states are like HMM tags.

- Appropriate for phonology and morphology
  
  Word = Syllable+
  = (Onset Nucleus Coda?)+
  = (C+ V+ C*)+
  = ( (b|d|f|...) + (a|e|i|o|u) + (b|d|f|...)* ) +
How powerful are regexps / FSAs?

- But less powerful than CFGs / pushdown automata
  - Can’t do recursive center-embedding
  - Hmm, humans have trouble processing those constructions too ...

- This is the rat that ate the malt.
- This is the malt that the rat ate.

- This is the cat that bit the rat that ate the malt.
- This is the malt that the rat that the cat bit ate.

- This is the dog that chased the cat that bit the rat that ate the malt.
- This is the malt that [the rat that [the cat that [the dog chased] bit] ate].

finite-state can handle this pattern (can you write the regexp?)

but not this pattern, which requires a CFG
How powerful are regexps / FSAs?

- But less powerful than CFGs / pushdown automata
- More important: Less explanatory than CFGs
  - An CFG without recursive center-embedding can be converted into an equivalent FSA – but the FSA will usually be far larger
  - Because FSAs can’t reuse the same phrase type in different places

\[
S = \begin{cases} 
\text{Noun} \\
\text{Noun} \rightarrow \text{Verb} \rightarrow \text{Noun} \\
\text{duplicated structure} \\
\text{duplicated structure}
\end{cases}
\]

\[
NP = \begin{cases} 
\text{Noun} \\
\text{NP} \rightarrow \text{Verb} \rightarrow \text{NP} \\
\text{NP}
\end{cases}
\]
We’ve already used FSAs this way ...

- CFG with regular expression on the right-hand side:
  
  \[
  \begin{align*}
  X &\rightarrow (A \mid B) \ G \ H \ (P \mid Q) \\
  NP &\rightarrow (\text{Det} \mid \epsilon) \ Adj^* \ N
  \end{align*}
  \]

- So each nonterminal has a finite-state automaton, giving a “recursive transition network (RTN)”

\[
X \rightarrow \begin{array}{c}
\text{A} \\
\text{B} \\
\text{G} \\
\text{H} \\
\text{P} \\
\text{Q}
\end{array}
\]

Automaton state replaces dotted rule \((X \rightarrow A \ G \ H \ P)\)
We’ve already used FSAs once..

\[ \text{NP} \rightarrow \text{rules from the WSJ grammar become a single DFA} \]

\[
\begin{align*}
\text{NP} &\rightarrow \text{ADJP ADJP JJ JJ NN NNS} \\
&\quad | \text{ADJP DT NN} \\
&\quad | \text{ADJP JJ NN} \\
&\quad | \text{ADJP JJ NN NNS} \\
&\quad | \text{ADJP JJ NNS} \\
&\quad | \text{ADJP NN} \\
&\quad | \text{ADJP NN NN} \\
&\quad | \text{ADJP NN NNS} \\
&\quad | \text{ADJP NNS} \\
&\quad | \text{ADJP NPR} \\
&\quad | \text{ADJP NPRS} \\
&\quad | \text{DT} \\
&\quad | \text{DT ADJP} \\
&\quad | \text{DT ADJP JJ NN} \\
&\quad | \text{DT ADJP ADJP NN} \\
&\quad | \text{DT ADJP JJ JJ NN} \\
&\quad | \text{DT ADJP JJ NN} \\
&\quad | \text{DT ADJP JJ NN NN} \\
\end{align*}
\]

\text{etc.}
But where can we put our weights?

- CFG / RTN

- bigram model of words or tags (first-order Markov Model)

- Hidden Markov Model of words and tags together??
Another useful FSA ...

Wordlist

```
clear
clever
ear
ever
fat
father
```

Network

```
/ usr / dict / words
0.6 sec
25K words
206K chars
```

```
FSM
17728 states,
37100 arcs
```
Weights are useful here too!

Wordlist

<table>
<thead>
<tr>
<th>Word</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>0</td>
</tr>
<tr>
<td>clever</td>
<td>1</td>
</tr>
<tr>
<td>ear</td>
<td>2</td>
</tr>
<tr>
<td>ever</td>
<td>3</td>
</tr>
<tr>
<td>fat</td>
<td>4</td>
</tr>
<tr>
<td>father</td>
<td>5</td>
</tr>
</tbody>
</table>

Network

Computes a perfect hash!
Sum the weights along a word’s accepting path.
Example: Weighted acceptor

- Compute **number of paths** from each state *(Q: how?)*
- Successor states partition the path set
- Use offsets of successor states as arc weights
- **Q**: Would this work for an arbitrary numbering of the words?
Example: Unweighted transducer

\[ VP \text{[head=vouloir,...]} \]

\[ V\text{[head=vouloir, ..., tense=Present, num=SG, person=P3]} \]

| veut |

the problem of morphology ("word shape") - an area of linguistics
**Example: Unweighted transducer**

Finite-state transducer

\[
\text{VP} [\text{head}=\text{vouloir}, ...]
\]

\[
\text{V} [\text{head}=\text{vouloir}, \text{...}, \text{tense}=\text{Present}, \text{num}=\text{SG}, \text{person}=\text{P3}]
\]

\[
\text{veut}
\]

canonical form

inflection codes

the relevant path

inflected form
Example: Unweighted transducer

- Bidirectional: generation or analysis
- Compact and fast
- Xerox sells for about 20 languages including English, German, Dutch, French, Italian, Spanish, Portuguese, Finnish, Russian, Turkish, Japanese, ...
- Research systems for many other languages, including Arabic, Malay

Finite-state transducer

vouloir +Pres +Sing + P3

veut

The relevant path

canonical form

inflection codes

inflected form
What is a function?

- Formally, a set of \(<\text{input}, \text{output}>\) pairs where each \(\text{input} \in \text{domain}\), \(\text{output} \in \text{co-domain}\).
- Moreover, \(\forall x \in \text{domain}, \exists \text{ exactly one } y\) such that \(<x,y> \in \text{the function}\).

\[
\begin{align*}
\text{domain(\text{square})} &= \{0, 1, -1, 2, -2, 3, -3, \ldots\} \\
\text{range(\text{square})} &= \{0, 1, 4, 9, \ldots\}
\end{align*}
\]
What is a relation?

- **square**: \( \text{int} \rightarrow \text{int} \)
  
  \[
  = \{ <0,0>, <1,1>, <-1,1>, <2,4>, <-2,4>, <3,9>, <-3,9>, \ldots \}
  \]

- **inverse(square)**: \( \text{int} \rightarrow \text{int} \)
  
  \[
  = \{ <0,0>, <1,1>, <1,-1>, <4,2>, <4,-2>, <9,3>, <9,-3>, \ldots \}
  \]

- Is \(\text{inverse(square)}\) a function?
  - 0 \(\mapsto\) \(\{0\}\) 9 \(\mapsto\) \(\{3,-3\}\) 7 \(\mapsto\) \(\{\}\) -1 \(\mapsto\) \(\{\}\)

- No - we need a more general notion!
  - A **relation** is any set of \(<\text{input}, \text{output}>\) pairs
**Regular Relation (of strings)**

- Relation: like a function, but multiple outputs ok
- Regular: finite-state
- Transducer: automaton w/ outputs

- $b \rightarrow \{b\}$  $a \rightarrow \{\}$
- $aaaaa \rightarrow \{ac, aca, acab, acabc\}$

- Invertible?
- Closed under composition?
Regular Relation (of strings)

- Can weight the arcs: $\rightarrow$ vs. $\rightarrow$
- $b \rightarrow \{b\}$ $a \rightarrow \{\}$
- $aaaaa \rightarrow \{ac, aca, acab, acabc\}$

- How to find best outputs?
  - For $aaaaa$?
  - For all inputs at once?
Function from strings to ...

Acceptors (FSAs)

Unweighted

\[
\begin{align*}
& a : x / .5 \\
& \varepsilon : y / .3 \\
& \{\text{false, true}\} \\
& c : z
\end{align*}
\]

Weighted

\[
\begin{align*}
& a : x / .5 \\
& \varepsilon : y / .5 \\
& \text{numbers} \\
& c : z / .7
\end{align*}
\]

Transducers (FSTs)

\[
\begin{align*}
& \text{strings} \\
& a : x \\
& \varepsilon : y \\
& c : z
\end{align*}
\]

\[
\begin{align*}
& a : x / .5 \\
& \varepsilon : y / .5 \\
& \text{(string, num) pairs} \\
& c : z / .7
\end{align*}
\]
Sample functions

Acceptors (FSAs)

Unweighted

\{false, true\}

Grammatical?

Weighted

numbers

How grammatical?
Better, how probable?

Transducers (FSTs)

strings

Markup
Correction
Translation

(string, num) pairs

Good markups
Good corrections
Good translations
Terminology (acceptors)

Regular language

- defines
- compiles into
- implements
- recognizes
- accepts (or generates)

Regexp

String

FSA

String matches Regexp and FSA accepts (or generates) it. Regular language defines Regexp and compiles into FSA.
Terminology (transducers)

Regular relation

- defines
- compiles into
- implements

Regexp

- matches

String pair

- matches
- accepts (or generates) (or transduces one string into the other)

FST

- recognizes

?
Perspectives on a Transducer

- Remember these CFG perspectives:

  3 views of a context-free rule
  
  - generation (production): \( S \rightarrow NP\ VP \) (rand sent)
  - parsing (comprehension): \( S \leftarrow NP\ VP \) (parse)
  - verification (checking): \( S = NP\ VP \)

- Similarly, 3 views of a transducer:
  - Given 0 strings, generate a new string pair (by picking a path)
  - Given one string (upper or lower), transduce it to the other kind
  - Given two strings (upper & lower), decide whether to accept the pair

FST just defines the regular relation (mathematical object: set of pairs). What’s “input” and “output” depends on what one asks about the relation. The 0, 1, or 2 given string(s) constrain which paths you can use.
Functions

\[ \text{ab?d} \rightarrow f \rightarrow \text{abcd} \rightarrow g \rightarrow \alpha \beta \chi \delta \]
Functions

Function composition: $f \circ g$

[first f, then g – intuitive notation, but opposite of the traditional math notation]

Like the Unix pipe: `cat x | f | g > y`
Example: Pass the input through a sequence of ciphers
From Functions to Relations

\[ f\left(\text{ab?d}\right) \rightarrow \text{abcd} \rightarrow \text{abed} \rightarrow \text{abjd} \quad \text{g}\left(\text{abcd}\right) \rightarrow \alpha \beta \chi \delta \rightarrow \alpha \beta \varepsilon \delta \rightarrow \alpha \beta \delta \rightarrow \ldots \]
From Functions to Relations

Relation composition: $f \circ g$

ab?d → 3 → 6

2 → 4 → α β χ δ

2 → α β ε δ

8 → α βδ

...
From Functions to Relations

Relation composition: \( f \circ g \)
From Functions to Relations

Often in NLP, all of the functions or relations involved can be described as finite-state machines, and manipulated using standard algorithms.
Inverting Relations

\[ \text{ab?d} \xrightarrow{f} \text{abcd} \xrightarrow{g} \text{abcd} \]

\[ \xrightarrow{2} \text{abed} \]

\[ \xrightarrow{6} \text{abjd} \]

\[ \xrightarrow{3} \]

\[ \xrightarrow{4} \]

\[ \xrightarrow{8} \]

\[ \ldots \]
Inverting Relations

ab?d \rightarrow f^{-1} \rightarrow 3 \rightarrow abcd \rightarrow g^{-1} \rightarrow \alpha^4 \beta \chi \delta

2 \rightarrow abed \rightarrow \alpha^2 \beta \epsilon \delta

6 \rightarrow abjd \rightarrow \alpha \beta \delta

...
Inverting Relations

\[(f \circ g)^{-1} = g^{-1} \circ f^{-1}\]
Building a lexical transducer

big | clear | clever | ear | fat | ...

Regular Expression Lexicon → Lexicon FSA → Lexical Transducer (a single FST)

Compiler

Regular Expressions for Rules → Composed Rule FSTs

Composition

one path
Building a lexical transducer

- Actually, the lexicon must contain elements like `big +Adj +Comp`
- So write it as a more complicated expression:
  
  \[
  (\text{big} \mid \text{clear} \mid \text{clever} \mid \text{fat} \mid ...) +\text{Adj} (\varepsilon \mid +\text{Comp} \mid +\text{Sup}) \leftarrow \text{adjectives}
  \mid (\text{ear} \mid \text{father} \mid ...) +\text{Noun} (+\text{Sing} \mid +\text{Pl}) \leftarrow \text{nouns}
  \mid ...
  \]
- Q: Why do we need a lexicon at all?
Weighted version of transducer: Assigns a weight to each string pair

Weighted French Transducer

"upper language"

"lower language"

slide courtesy of L. Karttunen (modified)