Morphology with Finite-State Machines

Algorithms for NLP

September 18, 2014
Outline

1. Some linguistic phenomena
2. Difficulties with formally defining the problem of morphology
3. Attempts to use formal languages nonetheless

Recommended reading: Jurafsky & Martin, chapter 3.
## Regular Languages for NL Vocabularies

<table>
<thead>
<tr>
<th>Formal languages terminology</th>
<th>Natural language terminology</th>
<th>Instantiated for English</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphabet, $\Sigma$</td>
<td>orthographic symbols</td>
<td>{A, ..., Z, a, ..., z}</td>
</tr>
<tr>
<td>word</td>
<td>word</td>
<td>whitespace-separated token (more or less)</td>
</tr>
<tr>
<td>language</td>
<td>vocabulary</td>
<td>(at least) all the words in a corpus, dictionary</td>
</tr>
</tbody>
</table>
1. Morphological phenomena
Morphemes

• A morpheme is a minimal, meaning-bearing unit of language.
  – Too small (in English): 'p'
  – Too big: 'processing'

• In some languages (Chinese), words and morphemes are basically the same.

• In some languages (Czech, Turkish), most words are made up of several morphemes.

• English is in the middle.
Inflectional Morphology

• Change a word, usually to make it agree.
  – La vida loca
  – El hombre loco

• Pluralizing a noun:
  – cat becomes cats
  – finch becomes finches
  – mouse becomes mice (irregular)

• Third person singular of a verb:
  – (catch) the cat catches a mouse
  – (kill) the cat kills mice
  – (have) the cat has a snack (irregular)

• Other tenses:
  – (kill) the cat is killing the mouse
  – (kill) the cat killed the mouse
  – (catch) the cat caught the mouse (irregular)
Irregularity is Common, Especially for Common Words

• The verbs *be, have, do*:
  
  – *be*   *have*   *do*
  
  – *am*   *have*   *do*
  
  – *are*   *have*   *do*
  
  – *is*   *has*   *does*
  
  – *was*   *had*   *did*
  
  – *were*   *had*   *did*
  
  – *been*   *had*   *done*
  
  – *being*   *having*   *doing*
Non-English Moment

• Inflectional morphology in other languages?

Derivational Morphology

• Nominalization
  – digitize $\rightarrow$ digitization
  – code $\rightarrow$ coder

• Creation of adjectives
  – computation $\rightarrow$ computational
  – clue $\rightarrow$ clueless

• Discourse
  – oedipus $\rightarrow$ oedipus schmoedipus

• These changes are less productive; you can't use them on every verb (or noun).
Claim

• Claim: morphology in human languages is **finite state**.
  – Big successes in modeling “morphology” languages like Turkish and Finnish

• Some difficult phenomena
  – Reduplication
  – Circumfixation
  – Root and Template Morphology
2. Tricky questions
So What is a “Word” in English?

• cat/cats, computer/computerize, haven't/have_not, caught/catch

• Can we represent the set of all of the words in a language?
  – How big is that set?

• Can we map observed/surface words to canonicalized forms?
  – Tokenization, lemmatization, stemming, morphological analysis
Orthography vs. Morphology

• In NLP, these sort of run together, especially in English.

• Text vs. speech applications: different needs

• We'll take a practical view; sometimes we're solving orthography problems and sometimes morphology problems.
Claims

• Formal language theory doesn't directly answer questions about natural language.
  – How many words?
  – What is the best way to define “word”?

• But it can help us *model* the phenomena of interest.
Attempt 1: List
List of Words

• Space requirements
• Runtime to answer query, “is this word in the language?”
Attempt 2: Trie
Trie

• Words that share a prefix can be packed together.
• Final states can be augmented with an index.
  – Useful if we want to integerize text.
• Runtime?
• Space?
Demo

• Using pyfst to build a trie.
Trie: Pros and Cons

+ “Going all the way down” to the letters
  • Good for efficiency on a computer!
+ It's a DFSA
  • Lots of theoretical and algorithmic results come for free

– No encoding of *regularity* in the lexicon
  • This would make it easier to build, modify, and understand our model
  • Shared structure = shared analysis?
Attempt 3: FSA
Encoding a Lexicon with an FSA

• Let's separate the representation of stems from the other morphemes.

• **Morphotactics:** a model explaining where the bound morphemes go, relative to the stem and each other.

• High-level idea: morphemes are like beads on a string – something FSAs model really well.
English Nominals

- Overgenerates: *foxs*
- Separately, we encode a sublexicon for each type of morpheme:

<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>
“Compilation”

• Given the sublexicons and the morphotactic FSA, we can:
  – Represent each sublexicon with an FSA (trie)
  – “Plug in” the sublexicons to the morphotactic FSA by replacing each arc with the set of all appropriate morphemes.
    • “FSA surgery”
  – The result is an FSA.
English Adjectives

• Overgenerates: *unbig, unfast, oranger, smally*.

• Should probably separate adjective stems that can take *un-* or -*ly* from those that cannot.
English Derivational Morphology

- Note sneaky introduction of A, N, V, Adv ... this isn't really part of the word!
From Recognition to Analysis/Parsing
Important Development

• So far, we've just been using FSAs to represent the set of strings in the vocabulary.
• We'd like to go farther, mapping strings to deeper analyses: lemma or stem, word type, inflectional features, and so on.
• For this, we need to generalize finite-state automata.
Finite-State Transducers

- Think of an automaton that works with two tapes at the same time.
  - (FSMs only had one tape; we could envision them as acceptors/recognizers, or as generators.)
- The language is a “string-pair” language
- FSTs can be understood as reading or writing either or both tapes!
  - **Recognizer**: take a pair of strings and accept if the pair is in the string-pair language, reject if not.
  - **Generator**: output pairs of strings.
  - **Translator**: Read one string and write out a string. (This is how we will use FSTs for morphological parsing.)
  - **Set relator**: compute relations between sets of strings.
## Defining a Finite-State Transducer

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q$</td>
<td>finite set of states</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>finite input vocabulary</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>finite output vocabulary</td>
</tr>
<tr>
<td>$q_0 \in Q$</td>
<td>start state</td>
</tr>
<tr>
<td>$F \subseteq Q$</td>
<td>set of final states</td>
</tr>
<tr>
<td>$\delta : Q \times \Sigma^* \rightarrow 2^Q$</td>
<td>transition function; set of possible next states given current state and input sequence</td>
</tr>
<tr>
<td>$\sigma : Q \times \Sigma^* \rightarrow 2^{\Delta^*}$</td>
<td>output function; set of possible output strings given current state and input sequence</td>
</tr>
</tbody>
</table>
Example

Graph:
- \( q_0 \) (initial state)
- \( q_1 \)
- \( q_2 \) (final state)

Transitions:
- \( q_0 \) to \( q_1 \): \( a:a, b:b \)
- \( q_0 \) to \( q_0 \): \( a:a \)
- \( q_1 \) to \( q_2 \): \( b:a \)
- \( q_2 \) to \( q_0 \): \( a:a \), \( b:b \)

Notes:
- States: \( q_0, q_1, q_2 \)
- Edges: \( a:a, b:b \)
Moore or Mealy?

• This is more like Mealy.
• But here we allow *sequences* of symbols on arcs.
• I leave it as an exercise for you to show that this doesn't change the expressive power; you can compile these FSTs down to ones with at most one input and at most one output symbol per arc.
FSTs and Regular Relations

• “String-pair language” = set of *pairs* of strings.
  – Isomorphic to FSTs in the same way regular languages are isomorphic to FSAs.
• Projection: extract only input or output side.
  – Result of projection is an FSA!
• FSAs are FSTs (identity relation)
• Not closed under difference, complementation, or intersection.
• Closed under: union, inversion (switch input and output labels), **composition**.
FST Composition (+Demo)

- If $T_1$ maps from $I_1$ to $O_1$ and $T_2$ maps from $O_1$ to $O_2$, then $T_1 \circ T_2$ maps from $I_1$ to $O_2$.
- The resulting relation holds for $(x, z)$ if there exists some $y$ such that $T_1$'s relation holds for $(x, y)$ and $T_2$'s relation holds for $(y, z)$. 

![Figure 3.9](image-url) The composition of $[a:b]^+$ with $[b:c]^+$ to produce $[a:c]^+$. 
Determinism?

• FSTs are nondeterministic in general.
• Not all FSTs can be determinized!
• Sequential FSTs are deterministic on their input.
  – At any state, given each input symbol, there is at most one transition out.
  – Modification: $\delta : Q \times \Sigma \rightarrow Q$ and $\sigma : Q \times \Sigma \rightarrow \Delta^*$
  – Epsilons okay on output, but not the input.
• Generalizations to allow finite amount of ambiguity: $p$-subsequential FSTs.
Non-Determinism

• Actually desirable for NLP!
FSTs for Morphology

• A word is understood as a pair of strings: one string is the \textit{lexical} level, the other is the \textit{surface} (spelling as seen in real data).
  – E.g., \texttt{cat +N +Pl | cats}
  – The mapping need not be one-to-one!
  – \textbf{Optionality}: 1 lexical string to many surface strings
  – \textbf{Ambiguity}: 1 surface string to many lexical strings
    • (But not an obvious solution to \textit{resolving} ambiguity!)
• Parsing: mapping from surface to lexical level.
Key Points about Composition

• Composing two FSTs gives us another FST
• Because FSAs are a special case of FSTs, we can:
  – Compose an FSA with an FST
    • ("match this input")
  – Compose an FST with an FSA
    • ("match this output")
  – Compose an FSA with an FST and with an FSA
    • ("what are all the ways to get this output from this input?")
FSTs for Morphology

• A word is understood as a pair of strings: one string is the **lexical** level, the other is the **surface** (spelling as seen in real data).
  – E.g., `cat +N +Pl / cats`
  – The mapping need not be one-to-one!
  – Optionality: 1 lexical string to many surface strings
  – Ambiguity: 1 surface string to many lexical strings
    • (But not an obvious solution to *resolving* ambiguity!)

• Parsing: mapping from surface to lexical level.
English Nominals: FST Version

Figure 3.13 A schematic transducer for English nominal number inflection $T_{num}$. The symbols above each arc represent elements of the morphological parse in the lexical tape; the symbols below each arc represent the surface tape (or the intermediate tape, to be described later), using the morpheme-boundary symbol $^\wedge$ and word-boundary marker #. The labels on the arcs leaving $q_0$ are schematic, and need to be expanded by individual words in the lexicon.

<table>
<thead>
<tr>
<th>reg-noun</th>
<th>irreg-pl-noun</th>
<th>irreg-sg-noun</th>
</tr>
</thead>
<tbody>
<tr>
<td>fox</td>
<td>g o:e o:e s e</td>
<td>goose</td>
</tr>
<tr>
<td>cat</td>
<td>sheep</td>
<td>sheep</td>
</tr>
<tr>
<td>aardvark</td>
<td>m o:i u:e s:c e</td>
<td>mouse</td>
</tr>
</tbody>
</table>
What's It Doing?

• Some string pairs for which the relation holds:

  fox +N +Sg, fox#
  (f:f o:o x:x +N:ε +Sg:#)

  fox +N +Pl, fox^s#
  (f:f o:o x:x +N:ε +Pl:^s#)

  goose +N +Sg, goose#
  (g:g o:o o:o s:s: e:e +N:ε +Sg:#)

  goose +N +Pl, geese#
  (g:g o:e o:e s:s: e:e +N:ε +Pl:#)

• Not actually getting to surface strings yet ... (fox^s# ?)
Back to Morphology and Orthography

• The green strings (fox^s#) can be understood as an *intermediate* tape between the lexical (blue) level and the surface.

• We can use FSTs to represent spelling change rules:
  – single-letter consonants get doubled before *-ing* and *-ed* (begging)
  – silent e gets dropped before *-ing* and *-ed* (making)
  – e gets inserted after -*s*, -*z*, -*x*, -*ch*, -*sh* and before s (watches)
  – -*y* gets changed to *-ie* before -*s*, -*i*, before *-ed* (tries)
  – verbs ending with vowel + c add -k (panicked)
Three Tapes

<table>
<thead>
<tr>
<th>f</th>
<th>o</th>
<th>x</th>
<th>+N</th>
<th>+Pl</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>o</td>
<td>x</td>
<td>^</td>
<td>s</td>
</tr>
<tr>
<td>f</td>
<td>o</td>
<td>x</td>
<td>e</td>
<td>s</td>
</tr>
</tbody>
</table>

lexical transducer is between these two tapes

orthographic transducer is between these two tapes
E Insertion as an FST

Figure 3.17 The transducer for the E-insertion rule of (3.4), extended from a similar transducer in Antworth (1990). We additionally need to delete the # symbol from the surface string; this can be done either by interpreting the symbol # as the pair #:ε, or by postprocessing the output to remove word boundaries.
Things to Notice

• “Insert an e on the red tape just when the intermediate tape has a morpheme ending in \( x \) (or \( z \), etc.) and the next morpheme is \( –s \).”

• The FST expresses only the constraints necessary for the rule.

• Other strings of symbols must pass through unchanged!
  – “other” symbol is syntactic sugar to help with this

• Strings that apply the rule when it shouldn't be applied need to be rejected!
Putting It All Together

• We have a lexicon transducer that maps between disambiguated forms and intermediate forms (blue/green).

• We have a bunch of orthography transducers that map between intermediate forms and weird English spelling (green/red).

• Amazing thing: we can combine these sensibly and end up with a single FST!
Parsing and Generation

Figure 3.19: Generating or parsing with FST lexicon and rules
Cascades

• Cascading: feeding the output of one transducer in as input to another.

• We can mechanically “fuse” the two transducers together – through composition – to get a single transducer that never explicitly represents the intermediate tape.

Crucial thing: FSTs are closed under composition!
Applying Rules In Parallel

• More than one orthographic rule might apply to the same word, so we don't want to cascade them.
• Since all rules are constructed to leave strings unchanged that they don't apply to, we can imagine applying the rules in parallel.
• **Intersection** is what we want.
  – But FSTs are not closed under intersection!
  – If strings are always of equal length, we're okay.
  – So treat $\varepsilon$ as a standard symbol when intersecting FSTs.
Composition and Intersection

Figure 3.21  Intersection and composition of transducers.
Toward A Parsing Algorithm

• In general, our FSTs will not be deterministic in any sense.
  – Claim: finding the set of valid outputs for a given input is extremely similar to the recognition algorithm for FSAs; just need to do more bookkeeping.
  – As we saw last time, this is a special case of composition.
Some Related Ideas

- Stemming (e.g., Porter's 1980 stemmer)
- Tokenization (English, Chinese)
- Spelling correction

- Hacking regular expressions in Perl
- Masterful use of grep

- Information extraction (e.g., FASTUS)
Remarks

• FSTs can be understood as a flexible, high-level, declarative programming language for working with string relations and sets.

• They can't do everything! But they are a powerful tool for certain kinds of jobs.

• There are nice implementations of FST algorithms, so you can focus on constructing the intuitive modules, then put them together using standard operations.
  – XFST, FOMA, OpenFST, PyFST, Thrax
Further Reading


• Roark and Sproat (2007): the first half covers lots of morphological phenomena and how they can be handled with FSMs.