**Motivation**

- Factoring words
  - Cats $\rightarrow$ CAT + N(oun) + PL(ural)

- Used in:
  - Traditional NLP applications
  - Finding word boundaries (e.g., Latin, Chinese)
  - Document retrieval (keyword retrieval)
  - Text classification
  - ...

**Morphology**

- Morphology is the study of how words are built up from smaller meaningful units called morphemes
  - Ex: disadvantages $\rightarrow$ dis + advantage + s

- 2 types:
  - Inflectional morphology (Hình thái học biến tố)
  - Derivational morphology (Hình thái học dẫn xuất)

**Inflectional morphology**

- A stem + a grammatical morpheme $\rightarrow$ a word:
  - the same class as the stem
  - relates to the syntax of a sentence

- Example: subject-verb agreement
  - He *hit*-s the ball
  - We *hit* the ball

- Plural and possessive markers
  - Cats, cat's
**Derivational morphology**

- **a stem + a grammatical morpheme → a word:**
  - different class, e.g., *transmit*→*transmission* (Verb to Noun)
  - Irregular meaning change

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Base Verb/Adjective</th>
<th>Derived form</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ation</td>
<td>computerize(V)</td>
<td>computerization(N)</td>
</tr>
<tr>
<td>-ee</td>
<td>appoint(V)</td>
<td>appointee(N)</td>
</tr>
<tr>
<td>-er</td>
<td>love(V)</td>
<td>lover(N)</td>
</tr>
<tr>
<td>-ness</td>
<td>fuzzy(Adj)</td>
<td>fuzziness</td>
</tr>
<tr>
<td>-less</td>
<td>clue(N)</td>
<td>clueless</td>
</tr>
</tbody>
</table>

**Problem**

Build a morphological parser to compute the morphology of words:

<table>
<thead>
<tr>
<th>Input</th>
<th>Morphological Parsed Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cats</td>
<td>Cat + N + PL</td>
</tr>
<tr>
<td>Cat</td>
<td>Cat + N + SG</td>
</tr>
<tr>
<td>Cities</td>
<td>City + N + PL</td>
</tr>
<tr>
<td>Goose</td>
<td>Goose + N + SG</td>
</tr>
<tr>
<td>Geese</td>
<td>Goose + N + PL</td>
</tr>
<tr>
<td>Gooses</td>
<td>Goose + V + 3SG</td>
</tr>
<tr>
<td>Merging</td>
<td>Merge + V + PRES-PART</td>
</tr>
<tr>
<td>caught</td>
<td>(catch + V + PAST-PART) or (catch + V + PAST-PART)</td>
</tr>
</tbody>
</table>

**Solution 1: A large dictionary**

Impractical: some languages associate a single meaning with a number of distinct surface forms (600 billion in Turkish)

German:

*Leben+s+versichert+gesellschaft+s+angestellter*  
(life+CmpAug+insurance+CmpAug+company+Comp Aug+employee)

Chinese compounding: about 3000 ‘words,’ combine to yield tens of thousands

**Solution 2: Look individual morphemes up**

- mis + interpret + ation + s  
  MIS + INTERPRET + noun form + plural

⇒ unrealistic: we might not find all the pieces in the dictionary, because of interference from the sound system (phonology)
Ex: cities ≠ citie + s; cities ≠ citi + es
Define the problem

What knowledge do we need?

- What endings follow what roots, and in what order
  - Cat/cats (*inflectional*)
  - Dog/dogged (*derivational*)
- Only some endings go on some words, not others
  - Do+er ok; (a class of verbs) but not following be
- **Spelling change** rules adjust the surface form vs. the lexicon form:
  - Get+er → double the t → getter
  - Fox+s → insert e → foxes
  - Fly+s → insert e → flies → Y to / → flies

Basic Terminology & Motivation

- **Stem**: core meaning unit (morpheme) of a word
- **Affixes**: pieces that combine with the stem to modify its meaning and grammatical functions
  - Prefix: *un-*, *anti-*, etc.
  - Suffix: *-ity*, *-ation*, etc.
  - Infix:
    - Tagalog: um+thinigi → humiring (borrow)

How to do?

- We want to model pure concatenation
- We need to ‘remember’ that certain items can only combine with certain other items
- There’s a perfect model for this – finite-state automata

Picture of finite-state automata (fsa):
## How: 2-level machine

**Surface form**

- f l i e s

**Finite-state transducer**

**Lexicon**

**Underlying form**

- F L Y + S

---

## Definition of finite-state automaton (fsa)

- **A (deterministic) finite-state automaton** (FSA) is a quintuple \((Q, \Sigma, q_0, F)\) where
  - \(Q\) is a finite set of states
  - \(\Sigma\) is a finite set of terminal symbols, the alphabet
  - \(q_0 \in Q\) is the initial state
  - \(F \subseteq Q\), the set of final states
  - \(\delta\) is a function from \(Q \times \Sigma\) into \(Q\), the transition function

---

## Formal languages & grammars

- **A language** is a set of strings defined over some alphabet \(\Sigma\), with some properties:
  
  \[ L = \{ x \in \Sigma^* \mid P(x) \} \]

- Suppose \(\Sigma = \{a, b\}\). Then we can have:
  
  - \(I_1 = \{a, aa, aaa\}\)
  - \(I_2 = \{a, aa, aaa, \ldots\}\)
  - \(I_3 = \{ba, baa, baaa, \ldots\}\)
  - \(I_4 = \{x \in \{a, b\}^* \mid a^n b^n, n \geq 0\}\)

---

## Plan:

1. Build fsa to recognize different stem-endings and prefix-stems
2. Build fsa to recognize spelling changes
3. Turn these into parsers by turning the fsa’s into finite-state transducers
Using fsa’s to build recognizer for morphophonemic forms

1. Build fsa system for English *inflectional morphology*
2. English *derivational morphology* fsa
3. Use this to *recognize* a valid word
4. Then show how to *parse* by extended to *transducer*
5. Add spelling-change rules

FSA for nominal inflection

- Remember, we *don't* have to worry about spelling changes
- 2 classes of word:
  - Regular: cat, table, city: add *s*
  - Irregular: goose, mouse, sheep (memorize)

Resulting fsa

English derivational morphology

- Much more complex than *inflectional*
- Consider adjectives:
  - *Big*, bigger, biggest
  - *Cool*, cooler, coolest, cooly
  - *Clear*, clearer, clearest, clearly, unclear, unclearly
  - *Happy*, happier, happiest, happily
  - *Unhappy*, unhappier, unhappiest, unhappily
Will this fsa work?

- Accepts all adjectives above, but
- Also accepts *unbig, realest*
- Common problem: *overgeneration*
- Solution?

Need *classes* of roots that say *which* can occur with which suffixes

Revised picture

More English
FSA at the level of individual letters

Aardvarks, foxs, ...

From recognizer to transducer

- Why: need to map (correspond) inputs and outputs (e.g., goose-geese)
- A finite state transducer is a quintuple:
  - $Q$: a finite set of states;
  - $\Sigma$: a finite alphabet of complex symbols. Each is an input-output pair, $i:o$, from alphabet $I$ and $o$ from alphabet $O$. So $\Sigma \subseteq I \times O$. $I,O$ can include the empty symbol $\epsilon$ or $\lambda$;
  - $q_0$: a start state;
  - $F$: the set of final states, $F \subseteq Q$;
  - $\delta$: the transition function between states.

FSA vs. FST

- An FSA defines a formal language (a set of strings)
- An FST defines a relation between sets of strings (defines a set of pairs of strings)

FSTs in morphological processing

2 operations

- Composition ($T_1 \circ T_2$): if transducer $T_1$ maps from $I_1$ to $O_1$, and $T_2$ from $I_2$ to $O_2$, then $T_1 \circ T_2$ maps from $I_1$ to $O_2$.
  - Useful to replace series of transducers

- Inversion ($T^{-1}$): $T(T^{-1})$ switches input and output labels.
  - Useful to convert parser to generator.
Automaton for singular/plural suffix, call this $T_{num}$

$(\text{cats#}, \text{cat N PL})$

$T_{lex} = T_{num} \circ T_{stems}$

Automaton for stems, call this $T_{stem}$

Spelling change rules

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consonant Doubling</td>
<td>1-letter consonant doubled before -ing/ed</td>
<td>beg/begging</td>
</tr>
<tr>
<td>E deletion</td>
<td>Silent e dropped before -ing, -ed</td>
<td>make/making</td>
</tr>
<tr>
<td>E insertion</td>
<td>e added after -s, -z, -ch, -sh before -s</td>
<td>fox/foxes</td>
</tr>
<tr>
<td>Y replacement</td>
<td>-y changes to -ie before -ed</td>
<td>try/tries</td>
</tr>
<tr>
<td>I spelling (l)</td>
<td>I goes to y before vowel</td>
<td>lie/lying</td>
</tr>
</tbody>
</table>
Fst spelling of “foxes” ↔ “FOX+S”

So another view of the situation:

is this (see notes2)

Recognizing ‘foxes’

root= always 1st class

root= always 1st class

33

Two-level morphology parsing (analysis) algorithm

1. Initialize set of paths to \( P = \{} \).
2. Read input symbols, one at a time.
3. At each symbol, generate all lexical symbols possibly corresponding to the \( \ast \) (empty) symbol.
4. Prolong all paths in \( P \) by all such possible \( (x:0) \) pairs.
5. Check each new path extension against the phonological FST and lexical FSA (lexical symbols only); delete impossible paths prefixes.
6. Repeat 4-5 until max. \# of consecutive \( 0 \)s reached.

Parsing Algorithm, cont’d

7. Generate all possible lexical symbols (get from all FSTs) for the current input symbol, form pairs.
8. Extend all paths from \( P \) using all such pairs.
9. Check all paths from \( P \) (next step in FST/FSA). Delete all outright impossible paths.
10. Repeat from 3 until end of input.
11. Collect lexical “glosses” from all surviving paths.
**Generation algorithm**

- Do not use the lexicon (well you have to put the “right” lexical strings together somehow!)
- Start with a lexical string \(L\).
- Generate all possible pairs \(l:s\) for every symbol in \(L\).
- Find all (hopefully only 1!) traversals through the FST which end in a final state.
- From all such traversals, print out the sequence of surface letters.