Morphology, Finite State Machines, and Regular Expressions

CS114 Lab 2
January 26, 2018
Kenneth Lai

Morphology

- Morphology—how to build words out of morphemes
- Morphemes—building blocks for words
  - Stems—”main” morpheme, “main” meaning
  - Affixes—modify the stem

Morphology

- Stems vs. affixes
  - Cat -s
  - Un- believe -able -ly
  - Dog

Morphology

- Inflectional morphology—word similar to stem
  - Same class (part of speech)
  - Similar meaning
- Derivational morphology—word not as similar to stem
  - Different class (part of speech)
  - Not as similar meaning

Morphology

- Inflectional morphology
  - Compute + ing = Computing
- Derivational morphology
  - Compute + er = Computer

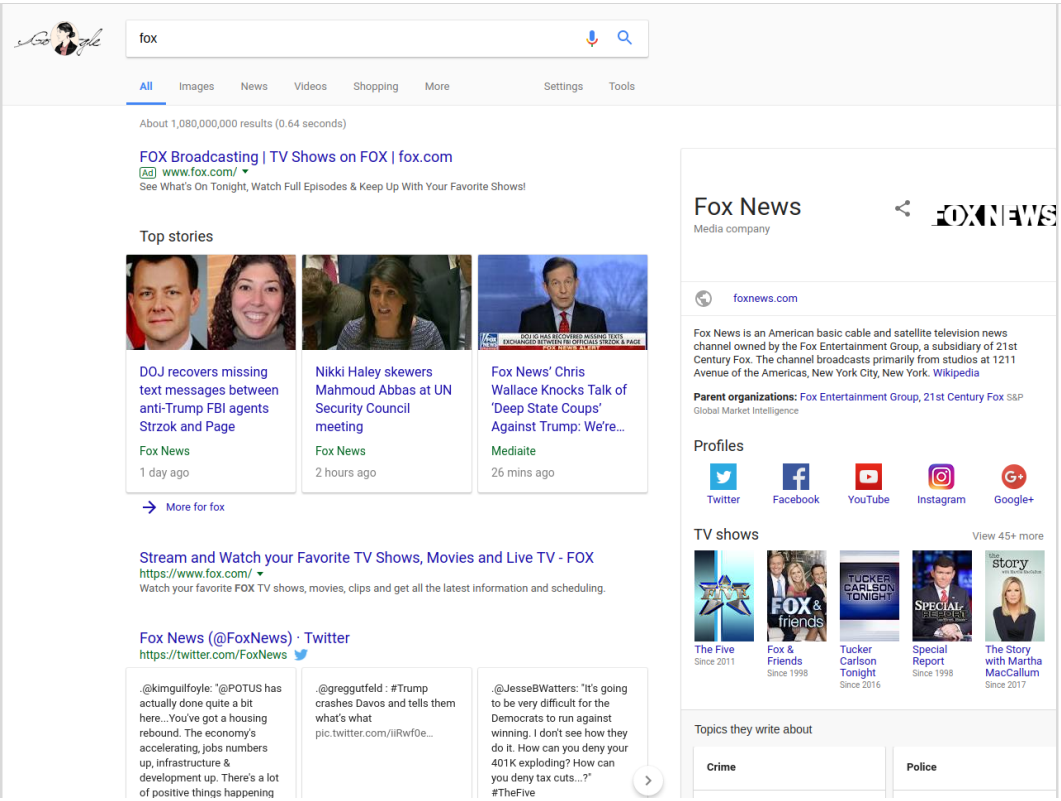
Morphology

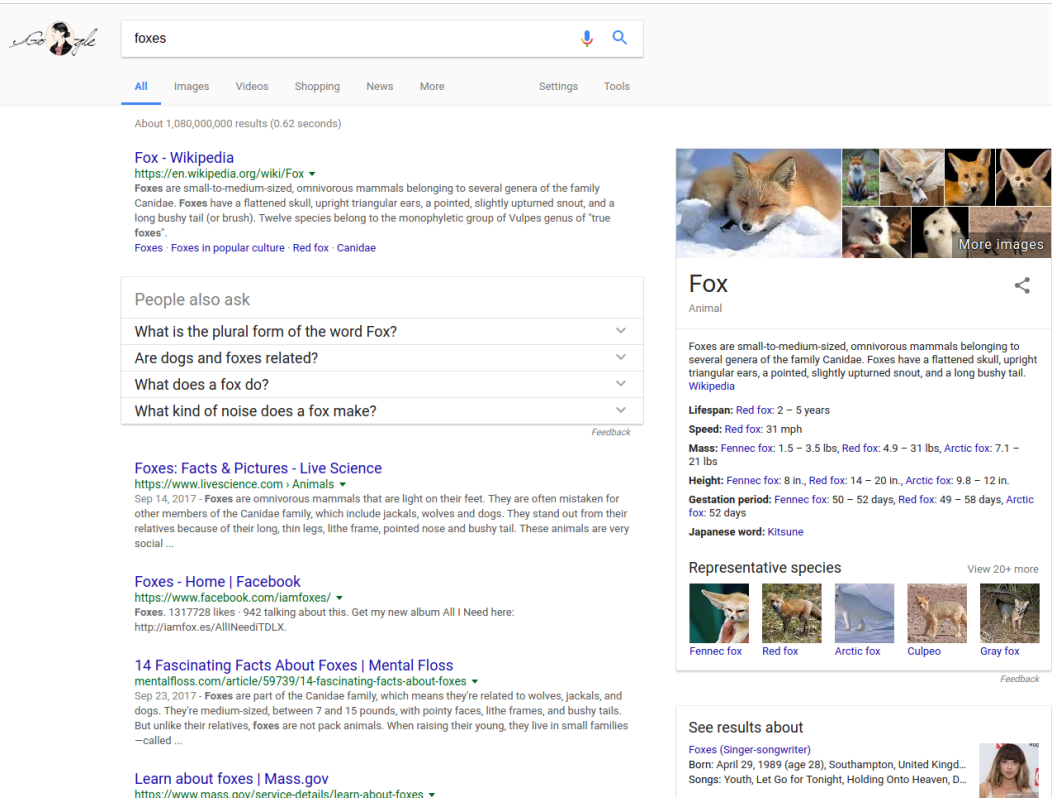
- Morphological parsing (HW problem 3)
  - Computing = Compute + ing = Compute + present participle
- Stemming
  - Computing = Compute

Uses of Morphology

- Information Retrieval
  - Example from book: Foxes = Fox

Uses of Morphology





Uses of Morphology

- Machine translation (and natural language generation more generally)
- Loves = love + s = love + 3SG = ama

	Present Indicative	Imperfect Indicative	Future	Preterite	Present Subjunctive	Conditional	Imperfect Subjunctive	Future Subjunctive
1SG	amo	amaba	amaré	amé	ame	amaría	amara	amare
2SG	amas	amabas	amarás	amaste	ames	amarías	amaras	amares
3SG	ama	amaba	amará	amó	ame	amaría	amara	amáreme
1PL	amamos	amábamos	amaremos	amamos	amemos	amaríamos	amáramos	amáremos
2PL	amáis	amabais	amaréis	amasteis	améis	amaríais	amarais	amareis
3PL	aman	amaban	amarán	amaron	amen	amarían	amaran	amaren

Figure 3.1 To love in Spanish. Some of the inflected forms of the verb *amar* in European Spanish. *1SG* stands for "first person singular", 3PL for "third person plural", and so on.

Finite State Machines

- Computers with almost no memory
- Useful for morphology (but not syntax)

Finite State Machines

- Automata
  - Output: just accept or reject
- Transducers
  - Output: an output string (or anything you want)

Finite State Machines

- Input: a string (containing symbols from some alphabet)
  - Read one symbol at a time from left to right
- Start in the start (initial) state
- Transition to next state according to current state and read symbol

Finite State Machines

- If you get to the end of the input and you are in:
  - Accept (final) state: return “accept”/output string
  - Other state: return “reject”/do not output string

Finite State Machines

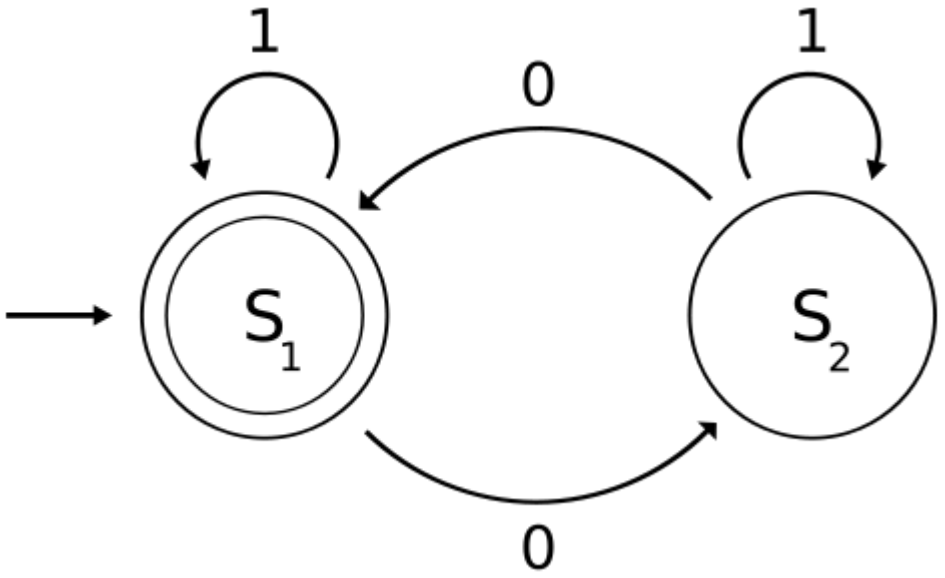
- Determinism—exactly one transition for each combination of state/symbol
- Nondeterminism—zero or more transitions for each combination of state/symbol

Finite State Machines

- For transducers, distinguish between nondeterministic transducers and nondeterministic transductions
  - Transducer—as before
  - Transduction—more than one possible output string for any input string
  - For HW, make sure all transductions are deterministic

Finite State Machines

- State diagram



Finite State Machines

- Language recognized by a finite state machine
  - Set of input strings for which the machine returns “accept”/output string

- A language is regular if and only if it is recognized by a finite state machine

Regular Expressions

- In theory
  - A sequence of characters that defines a regular language
- In practice
  - A sequence of characters that defines a search pattern (not necessarily regular)

Regular Expressions

- Base cases
  - The empty set ∅
  - The empty string ε
  - A symbol from the alphabet

Regular Expressions

- Regular operations
  - Union
    - a|b
  - Concatenation
    - ab
  - Kleene star
    - a*

Regular Expressions

- Other things you might see
  - [A-Za-z]
  - [^a]
  - a?
  - a+
  - \d
  - \1
- Some (but not all) of these are syntactic sugar

Regular Expressions in Python

- NLTK book
- Also: re in the Python Standard Library