Speech Acts

- Since the 1950s (Wittgenstein), communication has been seen as a set of speech acts.
- Communication as a form of action.
- Acts include: query, inform, request, acknowledge, promise.
- An agent has a goal that it needs to accomplish, and selects speech acts that help it to accomplish that goal.

Speech Acts

- An agent has a speech act it wants to achieve.
- It must convert this, plus some internal knowledge, into an utterance in a particular language.
- This utterance is then transmitted to a hearer or receiver.
- The hearer must then translate this back into an internal representation and reason about this new knowledge.

Language

- A language consists of a (possibly infinite) set of strings.
- These strings are constructed through the concatenation of terminal symbols.
- We'll distinguish between formal languages and natural languages.
- Formal languages have strict mathematical definitions.
- We can unambiguously whether a string is a legal utterance in that language.
- SQL, first-order logic, Java, and Python are all formal languages.

Natural Language

- Natural languages do not have a strict mathematical definition.
- They have evolved through a community of usage.
- English, Chinese, Japanese, Spanish, French, etc.
- Structure can be specified:
  - Prescriptively: What are the "correct" rules of the language.
  - Descriptively: How is the language actually used in practice?
- We'll attempt to treat natural languages as formal languages, even though the match is inexact.

Grammars

- A grammar is a set of rules that specifies the legal structure of a language.
- Each rule specifies how one or more symbols can be rewritten.
- Languages consist of terminal symbols, such as "cat", "the", "ran".
- These are our lexicon and nonterminal symbols, such as NP, VP, or S.
**Example Lexicon**
- Noun -> cat | dog | bunny | fish
- InTransVerb -> sit | sleep | eat
- TransVerb -> is
- Adjective -> happy | sad | tired
- Adverb -> happily | quietly
- Gerund -> sleeping
- Article -> the | a | an
- Conjunction -> and | or | but

**Example Grammar**
- S -> NP VP | S -> S Conjunction S
- NP -> Noun | Article Noun
- VP -> InTransVerb | TransVerb Adjective | InTransVerb Adverb | InTransVerb Gerund

**Syntax and Semantics**
The grammar of a language forms its **syntax**. This describes the structure of a sentence, and defines legal sentences.

The **semantics** of a sentence describes its actual meaning.
- This might be expressed in some sort of internal representation, such as SQL, logic, or a data structure.

The **pragmatics** of a sentence describes its meaning in the context of a given situation.
- “Class starts at 5:30” might have different meanings depending on the context.

**Classes of languages**
- We can characterize languages (and grammars) in terms of the strings that can be constructed from them.
- Regular languages contain rules of the form
  - $A \rightarrow b/A \rightarrow Bb$
  - Equivalent to regular expressions or finite state automata
  - Can’t represent (for example) balanced opening and closing parentheses.
- Context-free languages contain rules of the form
  - $A \rightarrow b/A \rightarrow XY$ (one nonterminal on left, anything on righthand side)
  - All programming languages are context free.
  - Natural languages are assumed to be context free.

**Classes of languages**
- Context-sensitive languages contain rules of the form
  - $ABC \rightarrow AQC$ (righthand side must contain at least as many symbols as left)
  - Some natural languages have context-sensitive constructs
  - Recursively-enumerable languages allow unrestricted rules.
  - They are equivalent to Turing machines.
  - We’ll focus on context-free grammars.

**Parsing**
- The first step in processing an utterance is **parsing**.
- Parsing is determining the syntactic structure of a sentence.
- Parts of speech and sentence structure
- This allows us to then try to assign meaning to components.
- Typically, parsing produces a **parse tree**.
Example

 Parsing as Search
- Parsing can be thought of as search
- Our search space is the space of all possible parse trees
- We can either start with the top of the tree and build down, or with the leaves and build up.

Top-down parsing
- Initial state: Tree consisting only of $S$.
- Successor function: Returns all trees that can be constructed by matching a rule in the grammar with the leftmost nonterminal node.
- Goal test: Leaves of tree correspond to input, no uncovered nonterminals.

Bottom-up parsing
- Bottom-up parsing takes the opposite approach:
- Start with leaves of the tree.
- Try to find right-hand sides of rules that match leaves.
- Work upward.
- Start state: A tree with leaves filled in.
- Successors: for each position in the tree, examine each rule, and return new trees by substituting right-hand sides for left-hand sides.
- Goal test: a tree with the root $S$.

Example

```
[S: ?]
[S: [NP: ?] [VP: ?]] - dead end - backtrack.
[S: [[Article: ?] [Noun: ?]] [VP: ?]]
[S: [[Article: The] [Noun: ?]] [VP: ?]] - dead end, backtrack.
[S: [[Article: The] [Noun: cat]] [VP: [IntransVerb: is] [Gerund: ?]]
[S: [[Article: The] [Noun: cat]] [VP: [IntransVerb: sleeping]]
```
Example

Init: 'The cat is sleeping' Succ: [[Art 'cat is sleeping'],
['the noun 'is sleeping' | 'the cat' InTransVerb 'sleeping']]
S1: [[Art 'cat is sleeping'] Succ: [[Art Noun 'is sleeping' | Art 'cat' InTransVerb 'sleeping' | Art 'is' Gerund]]
S2: [[Art Noun 'is sleeping' | Succ: [[NP 'is sleeping' | Art Noun 'is' Gerund]]]
S3: [NP 'is sleeping'] Succ: [NP IntransVerb 'sleeping'] [NP IntransVerb Gerund]
S4: [NP IntransVerb 'sleeping'] Succ: [NP IntransVerb Gerund]
S5: [NP IntransVerb Gerund] Succ: [NP VP]
S6: [NP VP] Succ: [S]

Bottom-up parsing

- While everything went fine in this simple example, there can be problems:
- Words might match multiple parts of speech
- The same right-hand side can match many left-hand sides
- Partial parses that could never lead to a complete sentence get expanded.

Efficient Parsing

- Consider the following sentences from R & N:
  - "Have the students in section 2 of CS 662 take the exam"
  - "Have the students in section 2 of CS 662 taken the exam?"
- If we parse this left-to-right (in a depth-first fashion) we can’t tell whether it’s a command or a question until we get to “take/taken”.
- We might then backtrack and have to rediscover that “the students in section 2 of CS 662” is an NP in either sentence.
- We need to keep track of partial results so that we don’t have to regenerate them each time.

Chart Parsing

- We keep track of the partial results of our parse in a data structure called a chart.
- The chart is represented as a graph. An n-word sentence produces a graph with n + 1 vertices representing each gap before, between, or after a word.
- Edges are added to the chart as parses are discovered for substrings.
- Edges are denoted with the starting and ending vertex, the parse discovered so far, and the parse needed to complete the string.

Example

- The edge from 0 to 2 is denoted with [S -> NP . VP]. This says that this edge matches an NP, and if you could find a VP, the sentence would be parsed.
- The edge from 2 to 4 is denoted with [VP -> Verb Gerund ] . This says that this substring contains a successful parse of a VP as Verb and Gerund.

Chart Parsing

- Chart parsing uses the best of both top-down and bottom-up methods:
- It starts top-down so as to take advantage of global structure
- It uses bottom-up only to extend existing partial parses
- It uses the chart to avoid repeated search
Chart Parsing

- Chart Parsing
  - chart(words)
    - for word in words:
      - scanner(word)
      - addEdge([0,0,S'-> .S])
    - scanner(word) foreach rule with word on the rhs:
      - add an edge
    - addEdge(edge):
      - if edge not in chart:
        - append edge to chart
      - if dot is at rhs:
        - extender(edge)
      - else:
        - predictor(edge)

Predictor([i,j, X -> a . B c]):
  - foreach rule that can predict B:
    - addEdge(rule)

Extender(rule):
  - foreach rule' with rule as part of the rhs:
    - replace rule' with rule

Example

- “The cat is sleeping”
  - Add an edge [0,0,S'->.S]
  - Since the . is not at the end, addEdge calls predictor to find a rule with S on the LHS.
  - Predictor adds the edge [0,0.0,S->. NP VP]
  - AddEdge calls predictor again to find a match for NP.
  - Predictor adds edges [0,0,NP->. Noun] and [0,0, NP->. Article Noun]
  - AddEdge then calls extender, which adds the edge [0,1, NP->. Article . Noun]

Example

- addEdge then calls extender, which adds the edge [0.2, NP -> Article Noun .]
- addEdge then calls extender to add the edge [0.2, S -> NP . VP]
- Extended adds the edge [2.3 VP -> InTransVerb . Adverb] and [2.3 VP -> InTransVerb . Gerund]
- Extender is called to add the edge [2.4, VP -> InTransVerb Gerund .]
- Extender is called to add the edge [S -> NP VP .] We have a successful parse.
Why are we doing this?

- So why go to all this effort?
- We want to determine the meaning (or semantics) of the sentence.
- Constructing a parse tree allows us to transform it into an internal representation that an agent can work with.
- Starting next week, we’ll look at logic as a representational form.

Challenges with NLP

- This is still a hard problem
- A sentence may have multiple parses: “Squad helps dog bite victim.”
- We need a complete lexicon for our language.
- Figures of speech, such as analogy, metaphor, and metonymy.
- “Apple announced a new iPhone this morning.”

Next time ...

- We’ll combine the statistical ideas from information theory with the structured approach of NLP.
- What’s the probability of a given parse?
- Speeding up parsing
- Dealing with incorrect grammar
- What meaning is the most likely?
- How should a phrase be segmented?