Artificial Intelligence Programming

Natural Language Processing

Cindi Thompson

Department of Computer Science

University of San Francisco

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 acomplish that goal.
- This lets us fit communication into our model of an agent choosing actions to accomplish a 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.

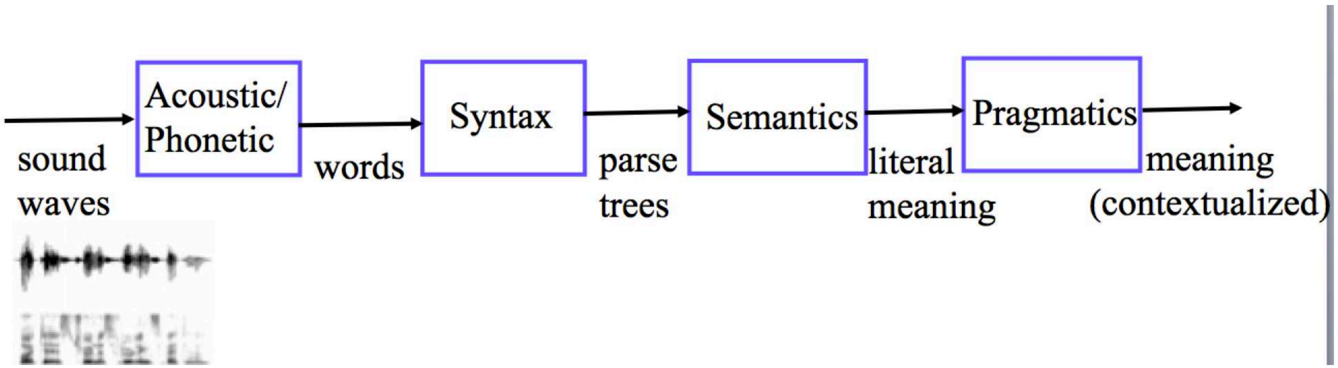
Language

- 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 say 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.

Processing Natural Language

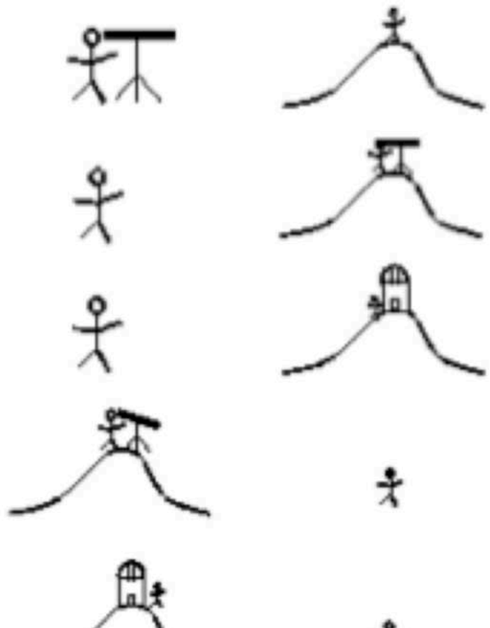


Ambiguity of NL

Natural language is highly ambiguous and must be disambiguated.

- I saw the man on the hill with a telescope.
- I saw the Grand Canyon flying to LA.
- Time flies like an arrow.
- Horse flies like a sugar cube.
- Time runners like a coach.
- Time cars like a Porsche.

Ambiguity



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 |
TransVerb 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 a database table, a set of logical sentences, or a data structure.
- The *pragmatics* of a sentence describes its meaning in the context of a given situation.
  - “Class starts at 3:30” 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

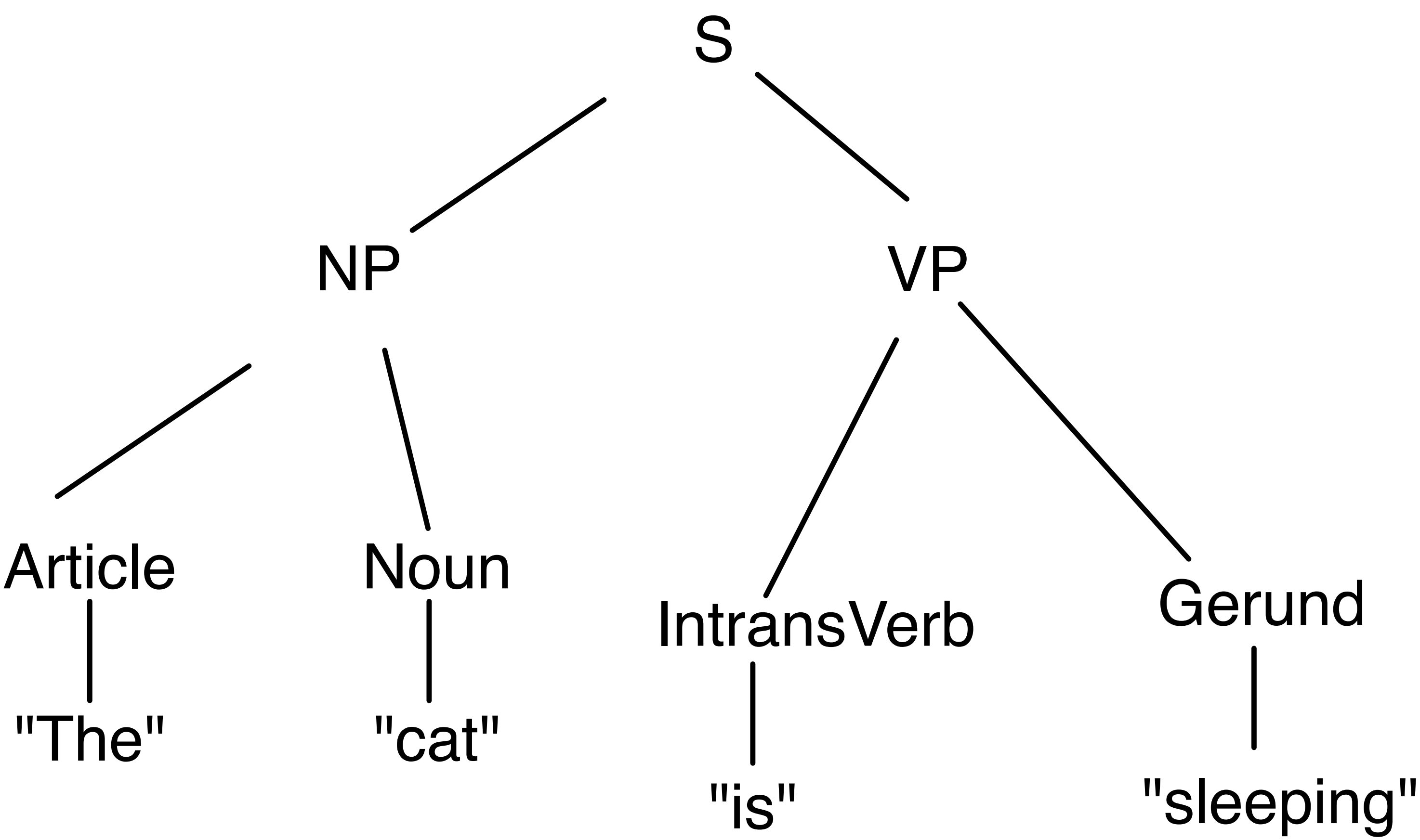
- Context-sensitive languages contain rules of the form $ABC \rightarrow AQC$ (righthand side must contain at least as many symbols as left)
  - Some natural languages have context-sensitive constructs
- Recursively enumerable languages allow unrestricted rules.
  - They are equivalent to Turing machines.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

"The cat is sleeping"



Note typo in POS for "is", should be TransVerb

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.

Top-down Example

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

Top-down parsing

- Top-down parsing has two significant weaknesses:
  - Doesn't exploit sentence structure at upper levels of the parse tree
    - Can wind up doing unnecessary search
  - Can't easily deal with left-recursive rules, such as $S \rightarrow S\ Conj\ S$
  - Can wind up infinitely re-expanding this rule, as in DFS.

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.

Bottom-up Example

```
Init: 'The cat is sleeping'
Succ: [[Art 'cat is sleeping'] , ['the' Noun 'is sleeping']
       ['the cat' TransVerb 'sleeping']]
S1: [Art 'cat is sleeping']
Succ: [[Art Noun 'is sleeping'] [Art 'cat' TransVerb 'sleeping']
       [Art 'cat is' Gerund]]
S2: [[Art Noun 'is sleeping']
Succ: [[NP 'is sleeping'] [Art Noun TransVerb 'sleeping']
       [Art Noun 'is' Gerund]]
S3: [NP 'is sleeping']
Succ: [[NP TransVerb 'sleeping'] [NP 'is' Gerund]]
S4: [NP TransVerb 'sleeping']
Succ: [NP TransVerb Gerund]
S5: [NP TransVerb 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 CS662” 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.

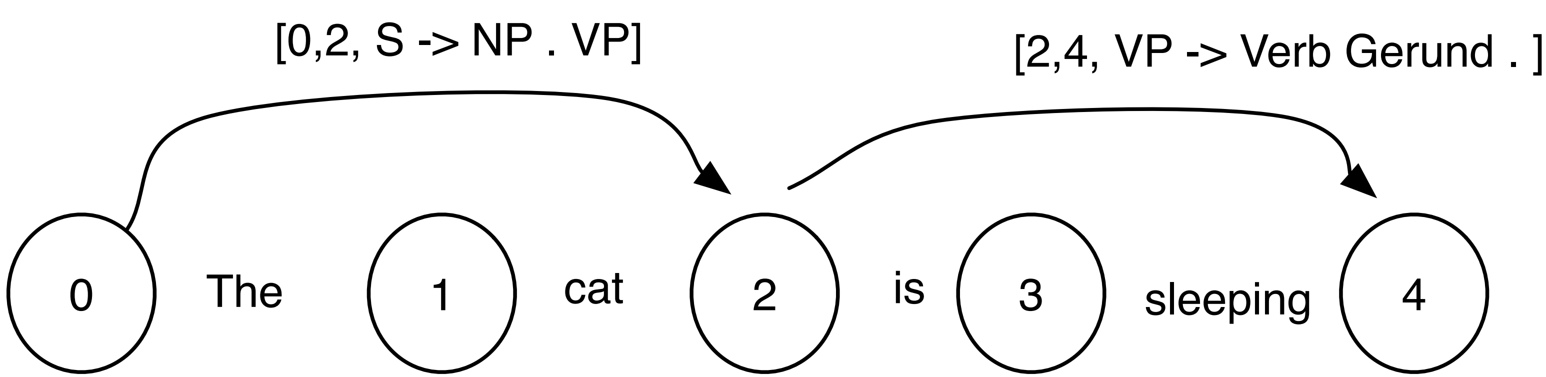


Chart Parsing

- 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

More on edges

The edges (stored in the Chart) are labeled with a "dotted" rule

- Indexed by start & end positions
- *inactive* edges are completed constituents: [VP -> Verb Gerund .]
- *active* edges are incomplete constituents: [S -> NP . VP] or [S -> . NP VP]

Chart and Agenda

The Agenda is like our search queue. The Chart keeps track of what we've done.

- Methods:
  - Init: Creates an agenda of edges to be processed
  - Rule invocation: Create an edge by matching an edge to a rule
  - Fundamental Rule: Create an edge by matching an inactive edge to an active edge

Chart parsing Algorithm

Top-down, breadth-first variant

```
chart = init(agenda, grammar, sentence)
while not agenda.isEmpty():
   edge = agenda.pop()
   if edge.isActive():
      agenda =
         RuleInvocation(edge, chart, grammar)
   edges = FundRule(edge, chart)
   agenda += edges
   chart.append(edge)
```

Initialization

Add an active edge for each S rule in your grammar

- `[0, 0, S -> . NP VP]`

Add an inactive edge for each word in sentence

- `[0, 1, the -> .]`
- `[1, 2, cat -> .]`
- ...

Rule Invocation

Given: an active edge [i, j, X -> a . B c]

- Find all rules with B on LHS
- Create new edges [j, j, B -> . P Q]
- Append to end of agenda

Note start and end positions!

Fundamental Rule

Match an *active* edge, from either chart or agenda, of the following form:

- `[i, j, X -> a . B c]`

With any matching *inactive* edges in the other data structure, of the form:

- `[j, k, B -> P Q .]`

Add to agenda:

`[i, k, X -> a B . c]`

What is the parse?

When the agenda is empty, each spanning edge in the Chart corresponds to a parse.

- `[i, j, S -> <parse> . ]`
- i = 0, j = 1+ len(sentence)
- ```
  [0, 4, S -> NP(Det(the) N(cat))
              VP(TransVerb(is) Gerund(sleeping))
  ```

There might've been more than one parse!
  ```

Chart Parsing

Advantages

- Allows found constituents to be reused
- Always terminates, even with left-recursive rules
- Does not generate (many) misleading sub-parses
- Uses $O(kn^2)$ space and $O(kn^3)$ time, n= sentence length, k depends on the grammar

Parsing is this hard?

Wait a minute. . . is parsing really this hard?

- Parsing is a key component of compiler design
- Thousand-line programs are parsed quite quickly
- No searching required

Parsing is this hard?

- Computer languages are very restricted
  - No ambiguity
  - Each “language fragment” always means the same thing
- We can create LL(1) grammars for most programming languages
  - Possible to look at the first token, and know which rule to apply
  - Take Programming Languages, or Compilers for much, much, more on parsing computer languages

Parsing is this hard?

- Natural Languages are *not* restricted at all
- No set way to encode any specific meaning
- Tremendously ambiguous
  - Even with unambiguous statements, often can't parse the first part of a sentence without seeing all the tokens
  - Context matters, too! (that's why people are pretty good at parsing)

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.
  - We could then map this parse tree to logical sentences in a KB.

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.”
    - “The White House supports the bill”