Computational Semantics

CS224N 2007
Christopher Manning

(Borrows some slides from Mary Dalrymple, Jason Eisner, and Jim Martin)

Why study computational semantics?

• Because everyone has been wanting me to talk about this all course!
• Obvious high-level applications
  • Summarization
  • Translation
  • Question answering
  • Information access
  • Talking to your pet robot
  • Speech user interfaces
• The next generation of intelligent applications need deeper semantics than we have seen so far
  • Often you must understand well to be able to act

Shallow vs. deep semantics

• We can do more than one might have thought without deep linguistic analysis
  • This is the lesson of the last decade
• But we can’t do everything we would like:
  • Not all tasks can ignore higher structure
  • Unsuitable if new text must be generated
  • Unsuitable if machine must act rather than relying on user to interpret material written by the author of the document
• You get what you pay for:
  • Cheap, fast, low-level techniques are appropriate in domains where speed and volume are more important than accuracy
  • More computationally expensive, higher-level techniques are appropriate when high-quality results are required

MSN Search: Which is the largest African country?

Live Search: Which is the largest African country?

Live Search: What is the capital of Sudan?
Precise semantics. An early example: Chat-80

- Developed between 1979 and 1982 by Fernando Pereira and David Warren; became Pereira’s dissertation
- Proof-of-concept natural language interface to database system
- Used in projects: e.g. Shoptalk (Cohen et al. 1989), a natural language and graphical interface for decision support in manufacturing
- Even used in an AppliedNLP-2000 conference paper [Asking about train routes and schedules]
- Available in cs224n src directory
  - Need sicstus prolog: /usr/sweet/bin/sicstus

The CHAT-80 Database

```prolog
% Facts about countries.
% country(Country,Region,Latitute,Longitude,
% Area (sqmiles), Population, Capital,Currency)
country(andorra,southern_europe,42,-1,179
  25000,andorra_la_villa,franc_peseta).
country(angola,southern_africa,-12,-18,481351,
  5810000,luanda,9).
country(argentina,south_america,-35,66, 1072067,
  23920000,buenos_aires,peso).
capital(C,Cap) :- country(C,_,_,_,_,_,Cap,_).
```
Question: What is the capital of Australia?

Answer: Canberra.

- How about $3+5\times x$?
- Don’t know $x$ at compile time
- “Meaning” at a node is a piece of code, not a number
- Form is “rule-to-rule” translation
  - We provide a way to form the semantics of each parent in terms of the semantics of the children

What Counts as Understanding?

- A somewhat difficult philosophical question
- We understand if we can respond appropriately
- “throw axe at dwarf”
- We understand statement if we can determine its truth
- We understand statement if we can use it to answer questions
- We assume we have logic-manipulating rules to tell us how to act, draw conclusions, answer questions ...

Lecture Plan

- Today:
  - Look at some sentences and phrases
  - What would be reasonable logical representations for them?
  - Get some idea of compositional semantics
  - An alternative semantic approach
  - Semantic grammars
- Next Wednesday:
  - How can we build those representations?
- Another course (somewhere in AI, hopefully):
  - How can we reason with those representations?
- Last week of lectures:
  - Lexical semantics
  - Question answering/semantic search/textual entailment

Logic: Some Preliminaries

Three major kinds of objects
1. Booleans (Bool)
   - Roughly, the semantic values of sentences
2. Individuals/Entities (Ind)
   - Values of NPs, i.e., objects
   - Maybe also other types of entities, like times
3. Functions of various types
   - A function returning a boolean is called a “predicate”
   - E.g., (frog, green)
   - A predicate defines a set of individuals that satisfy it
   - A one argument predicate is called a “property”
   - More complex functions return other functions!
   - Some functions take other functions as arguments!
   - (Higher order functions.)
Logic: Lambda Terms

- **Lambda terms:**
  - A way of writing "anonymous functions"
  - No function header or function name
  - But defines the key thing: behavior of the function
  - Just as we can talk about 3 without naming it "x"
- Let \(\text{square} = \lambda p. p^2\)
- Equivalent to \(\text{int square(p) \{ return p*p; \}}\)
- But we can talk about \(\lambda p \ p^2\) without naming it
- Format of a lambda term: \(\lambda \text{variable} \ . \text{expression}\)

Logic: Lambda Terms

- **Lambda terms:**
  - Let \(\text{square} = \lambda p. p^2\)
  - Then \(\text{square}(3) = \lambda p. p^2(3) = 3^2\)
  - Note: \(\text{square}(x)\) isn’t a function! It’s just the value \(x^2\).
  - But \(\lambda p {\text{square}}(p)\) and \(\lambda p \ p^2\) are equal (proving that these functions are equal – and indeed they are, as they act the same on all arguments: what is \(\text{square}(\text{square}(p))\)?)
  - Let \(\text{even} = \lambda p (p \mod 2 == 0)\) a predicate: returns true/false
  - \(\text{even}(x)\) is true if \(x\) is even
  - \(\lambda x \ \text{even}(\text{square}(x))\) is true of numbers with even squares
  - Just apply rules to get \(\lambda x (\text{even}(x^2)) = \lambda x (x^2 \mod 2 == 0)\)
  - This happens to denote the same predicate as \(\text{even}\) does

Logic: Multiple Arguments

- All lambda terms have one argument
- But we can fake multiple arguments ...
- Suppose we want to write \(\text{times}(5,6)\)
- Remember: \(\text{square}\) can be written as \(\lambda x.\text{square}(x)\)
- Similarly, \(\text{times}\) is equivalent to \(\lambda x. (\lambda y.\text{times}(x,y))\)
- Claim that \(\text{times}(5,6)\) means same as \(\text{times}(5,\text{square}(6))\)
  - \(\text{times}(5,6) = 5 \times 6 = 30\)
  - \(\text{times}(5,\text{square}(6)) = 5 \times \text{square}(6) = 5 \times 6^2 = 5 \times 6^2 = 5 \times 36 = 180\)
  - Referred to as "currying"

Logic: Interesting Constants

- We have “constants” that name some of the entities and functions (e.g., \(\text{times}\)):
  - GeorgeBush - an entity
  - red - a predicate on entities
    - holds of just the red entities: \(\text{red}(x)\) is true if \(x\) is red!
  - loves - a predicate on 2 entities
    - \(\text{loves}(\text{GeorgeBush}, \text{LauraBush})\)
    - Question: What does \(\text{loves}(\text{LauraBush})\) denote?
  - Constants used to define meanings of words
  - Meanings of phrases will be built from the constants

Logic: Interesting Constants

- **Generalized Quantifiers**
  - most – a predicate on 2 predicates on entities
    - \(\text{most}(\text{pig}, \text{big})\) = “most pigs are big”
    - Equivalently, \(\text{most}(\lambda x \ \text{pig}(x), \lambda x \ \text{big}(x))\)
    - Returns true if most of the things satisfying the first predicate also satisfy the second predicate
  - similarly for other quantifiers
    - \(\text{all}(\text{pig}, \text{big})\) (equivalent to \(\lambda x \ \text{pig}(x) \Rightarrow \text{big}(x)\))
    - \(\text{exists}(\text{pig}, \text{big})\) (equivalent to \(\lambda x \ \text{pig}(x) \land \text{big}(x)\))
    - Can even build complex quantifiers from English phrases:
      - “between 12 and 75”; “a majority of”; “all but the smallest 2”

Quantifier Order

- Groucho Marx celebrates quantifier order ambiguity:
  - In this country a woman gives birth every 15 min. Our job is to find that woman and stop her.
  - \(\exists \text{woman} (\forall 15 \text{min gives-birth-during(woman, 15min)})\)
  - \(\forall 15 \text{min} (\exists \text{woman gives-birth-during(15min, woman)})\)
  - Surprisingly, both are possible in natural language!
  - Which is the joke meaning?
    - (where it’s always the same woman)
We’ve discussed what semantic representations should look like.
But how do we get them from sentences???
First - parse to get a syntax tree.
Second - look up the semantics for each word.
Third - build the semantics for each constituent
   Work from the bottom up
   The syntax tree is a “recipe” for how to do it
Principle of Compositionality
The meaning of a whole is derived from the meanings of the parts, via composition rules

Compositional Semantics

A simple grammar of English
(in Definite Clause Grammar, DCG, form - as in Prolog)
sentence --> noun_phrase, verb_phrase.
noun_phrase --> proper_noun.
noun_phrase --> determiner, noun.
verb_phrase --> verb, noun_phrase.
Proper_noun --> [John]  verb --> [ate]
Proper_noun --> [Mary]  verb --> [kissed]
determiner --> [the]   noun --> [cake]
determiner --> [a]   noun --> [lion]

Extending the grammar to check number agreement between subjects and verbs

S --> NP(Num), VP(Num).
NP(Num) --> Proper_noun(Num).
NP(Num) --> det(Num), noun(Num).
VP(Num) --> verb(Num), noun_phrase(_).
Proper_noun(s) --> [Mary]. noun(s) --> [lion].
det(s) --> [the]. noun(p) --> [lions].
det(p) --> [the]. verb(s) --> [eats].
verb(p) --> [eat].

Parse tree with associated semantics

Sentence
    loves(john,mary)
  /       \
Noun Phrase
    john
    / \
Name
  john
  / \
“John”

Verb Phrase
    loves(x,y)
  /       \
Noun Phrase
    name
    / \
Name
  mary
  / \
“Mary”

A simple DCG grammar with semantics

sentence(SMeaning) --> noun_phrase(NPMeaning), verb_phrase(VPMeaning), {combine (NPMeaning, VPMeaning, SMeaning)}.
verb_phrase(VPMeaning) --> verb(Vmeaning), noun_phrase(NPMeaning), {combine (NPMeaning, VMeaning, VPMeaning)}.
noun_phrase (NPMeaning) --> name(NPMeaning).
name(john) --> [john]. verb(\lambda x.\lambda y.\lambda z.loves(x,y)) --> [loves].
name(mary) --> [mary]. verb(\lambda y.\lambda x.loves(x,y)) --> [loves].

Augmented CFG Rules

We can also accomplish this just by attaching semantic formation rules to our syntactic CFG rules
\[ A \to \alpha_1 \ldots \alpha_n \{ f(\alpha_1,sem,\ldots,\alpha_n,sem) \}\]
This should be read as the semantics we attach to A can be computed from some function applied to the semantics of A’s parts.
The functions/operations permitted in the semantic rules are restricted, falling into two classes
  • Pass the semantics of a daughter up unchanged to the mother
  • Apply (as a function) the semantics of one of the daughters of a node to the semantics of the other daughters
How do things get more complex?
(The former) GRE analytic section

- Six sculptures – C, D, E, F, G, H – are to be exhibited in rooms 1, 2, and 3 of an art gallery.
- Sculptures C and E may not be exhibited in the same room.
- Sculptures D and G must be exhibited in the same room.
- If sculptures E and F are exhibited in the same room, no other sculpture may be exhibited in that room.
- At least one sculpture must be exhibited in each room, and no more than three sculptures may be exhibited in any room.
- If sculpture D is exhibited in room 3 and sculptures E and F are exhibited in room 1, which of the following may be true?
  1. Sculpture C is exhibited in room 1.
  2. Sculpture H is exhibited in room 1.
  3. Sculpture G is exhibited in room 2.
  4. Sculptures C and H are exhibited in the same room.
  5. Sculptures G and F are exhibited in the same room.

Scope Needs to be Resolved!

At least one sculpture must be exhibited in each room.
The same sculpture in each room?
No more than three sculptures may be exhibited in any room.
Reading 1: For every room, there are no more than three sculptures exhibited in it.
Reading 2: Only three or less sculptures are exhibited (the rest are not shown).
Reading 3: Only a certain set of three or less sculptures may be exhibited in any room (for the other sculptures there are restrictions in allowable rooms).
- Some readings will be ruled out by being uninformative or by contradicting other statements.
- Otherwise we must be content with distributions over scope-resolved semantic forms.

Semantic Grammars

- A problem with traditional linguistic grammars is that they don’t necessarily reflect the semantics in a straightforward way.
- You can deal with this by...
  - Fighting with the grammar
    - Complex lambdas and complex terms, etc.
  - Rewriting the grammar to reflect the semantics
    - And in the process give up on some syntactic niceties
    - Known as “Semantic grammars”
    - Simple idea, dumb name

Semantic Grammar

- The term semantic grammar refers to the motivation for the grammar rules.
- The technology (plain CFG rules with a set of terminals) is the same as we’ve been using.
- The good thing about them is that you get exactly the same thing.
- The bad thing is that you need to develop a new grammar for each new domain.
- Typically used in conversational agents in constrained domains.
- Limited vocabulary
- Limited grammatical complexity
- Syntactic parsing can often produce all that’s needed for semantic interpretation even in the face of “ungrammatical” input – write fragment rules

Lifer Semantic Grammars

- Example domain—access to DB of US Navy ships:
  - $ \rightarrow \text{<present> the <attribute> of <ship>}$
  - $\text{<present>}$ \rightarrow what is | can you tell me
  - $\text{<attribute>}$ \rightarrow length | beam | class
  - $\text{<shipname>}$ \rightarrow the <shipname> | enterprise
  - $\text{<classname>}$ \rightarrow class ships
  - $\text{<classname>}$ \rightarrow Kitty hawk | Lafayette

- Example inputs recognized by above grammar:
  - What is the length of the Kennedy?  The Kittyhawk?
  - Many categories are not “true” syntactic categories
  - Words are recognized by their context rather than category (e.g. class)
  - Recognition is strongly directed
  - Strong direction useful for error detection and correction

Semantic Grammars Summary

- Advantages:
  - Efficient recognition of limited domain input
  - Absence of overall grammar allows pattern-matching possibilities for idioms, etc.
  - No separate interpretation phase
  - Strength of top-down constraints allows powerful ellipsis mechanisms
  - What is the length of the Kennedy?  The Kittyhawk?

- Disadvantages:
  - Different grammar required for each new domain
  - Lack of overall syntax can lead to “spotty” grammar coverage
  - E.g. fronting possessive in “<attribute> of <ship>” to “<ship>’s <attribute>” doesn’t imply fronting in “<rank> of <officer>”
  - Difficult to develop grammars past a certain size
  - Suffers from fragility