Computational Semantics

CS224N 2004
Christopher Manning

(Borrows 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 applications
  - Summarization
  - Translation
  - Question answering
  - Information access
  - Talking to your pet robot
  - Speech user interfaces
- The next generation of intelligent applications need deeper semantics

Shallow vs. deep semantics

- We want to get from syntax to meaning!
- We can do more than we would have thought without deep linguistic analysis
- 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 aim is not just to hand the user part of a document, relying on the author of the document and the user to make sense of the result

Example:

Same words, different query

Kennedy assassination: LeeHarvey Oswald assassinated JFK, was shot to death several days later by Jack Ruby

"Who did Oswald assassinate?"
answer: President Kennedy

"Who assassinated Oswald?"
answer: Jack Ruby

Who assassinated Oswald?

Who did Oswald assassinate?
Moral: You get what you pay for

- Cheap, fast, low-level techniques are useful, 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

What we say to dogs

Okay, Ginger! I've had it! You stay out of the garbage! Understand Ginger? Stay out of the garbage, or else!

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 (Camb et al., 1989), a natural language and graphical interface for decision support in manufacturing
- Even used in an ANLP-2000 conference paper!
- Available in src directory
  - Need: status protol sax2mml/galaxy/symbase.xml

The CHAT-80 Database

% Facts about countries.
% country(Country,Region,Latitude,Longitude,
% Area (sqmiles), Population, Capital,Currency)
country(Andorra,southern_europe,42.5,1,179,
250000, Andorra, la_vila, franc peseta).
country(Angola,southern_africa,-12,-18,481351,
5810000, luanda, $).
country(Argentina,south_america,-35,66,
1072067, 23920000, buenos_aires, peso).
capital(C,Cap) :- country(C, ... , Cap, ... ).

Chat-80 trace (Illegibly small)

Question: What is the capital of Australia?
Parse: 0.0 sec.

$VAR s
  np $in
  wh (x)
  verb (active, pres = fn, [], [pp])
  np $in
  np $in

np (head (dec (can)))
[ ]
  capital
  np (pres (1))
  np $in
  name (australia)
[ ]
  Semantics: 0.0 sec.
  answer (0): -
  capital (australia, 0)
  camera.
Getting semantics: programming language interpreter

- What is meaning of 3+5*6?
- First parse it into 3+(5*6)

More complex meanings

- How about 3+5*x?
- Don’t know x at compile time
- “Meaning” at a node is a piece of code, not a number

What counts as understanding?

- A somewhat difficult philosophical question
- We understand if we can respond appropriately
- “throw me a dwarf”
- We understand statement if we can determine its truth
- We understand statement if we can use it to answer questions (similar to above - requires reasoning)
- Easy: John ate pizza. What was eaten by John?
- Understanding is the ability to translate
  - English to Chinese? requires deep understanding?
  - English to logic? deeper - the definition we’ll use
  - All humans are mortal = \( \forall x \ (\text{human}(x) \rightarrow \text{mortal}(x)) \)
  - 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
- Wednesday:
  - How can we build those representations?
  - Another course (somewhere in AI, hopefully):
    - How can we reason with those representations?

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., (prop(x, y))
    - Functions might return other functions!
    - Function might take other functions as arguments!
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 square = \lambda p \cdot p \cdot p
- Equivalent to int square(p) { return p * p; }
- But we can talk about \lambda p \cdot p without naming it
- Format of a lambda term: \lambda variable . expression

Logic: Multiple Arguments

- All lambda terms have one argument
- But we can fake multiple arguments...
- Suppose we want to write times(5,6)
- Remember: square can be written as \lambda x . square(x)
- Similarly, times can be equivalent to \lambda x \cdot \lambda y . times(x,y)
- Claim that times(5,6) means same as times(5,6)
- times(5) = (\lambda x . \lambda y . times(x,y)) (5) = \lambda y . times(5,y)
  - If this function weren’t anonymous, what would we call it?
- times(5,6) = (\lambda y . times(5,y))(6) = times(5,6)
- Referred to as "currying"

Logic: Interesting Constants

- Most - a predicate on 2 predicates on entities
  - most(pig, big) = "most pigs are big"
  - Equivalently, most(\lambda x . pig(x), \lambda x . big(x))
  - Returns true if most of the things satisfying the first predicate also satisfy the second predicate
- Similarly for other quantifiers
  - all(pig, big) equivalent to \forall x . pig(x) \Rightarrow big(x)
  - exists(pig, big) equivalent to \exists x . pig(x) \land big(x)
- can even build complex quantifiers from English phrases:
  - "between 12 and 75"; "a majority of"; "all but the smallest 2"

Logic: Interesting Constants

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

Logic: Interesting Constants

- 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 woman (\forall 15 min gives birth during (woman, 15 min))
  - \forall 15 min (\exists woman gives birth during (15 min, woman))
- Surprisingly, both are possible in natural language!
- Which is the joke meaning? (where it’s always the same woman)

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 woman (\forall 15 min gives birth during (woman, 15 min))
  - \forall 15 min (\exists woman gives birth during (15 min, woman))
- Surprisingly, both are possible in natural language!
- Which is the joke meaning? (where it’s always the same woman)
Compositional Semantics

- 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

A simple grammar of English

(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 --> [bion]

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(s) --> [lions].
det(p) --> [the]. verb(s) --> [eats].
verb(p) --> [eat].

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).
noun(s) --> [john]. verb(\(\lambda x.\)jumps(x)) --> [jumps]
noun(s) --> [mary]. verb(\(\lambda x.\)loves(x,y)) --> [loves]

Parse tree with associated semantics

Sentence
  Noun Phrase john
    Name john
    "John"
  Verb
    Noun Phrase
      verb
      Noun Phrase
        x
        "loves"
        y
        "Mary"
        "Mary"

Augmented CFG Rules

- We can also accomplish this just by attaching semantic formation rules to our syntactic CFG rules
  \[ A \rightarrow \alpha_1 \ldots \alpha_n \{ f (\alpha_1, sem_1, \ldots, \alpha_n, sem_n) \} \]
- 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 to the semantics of the other daughters