Computational Semantics

CS224N 2009
Christopher Manning

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

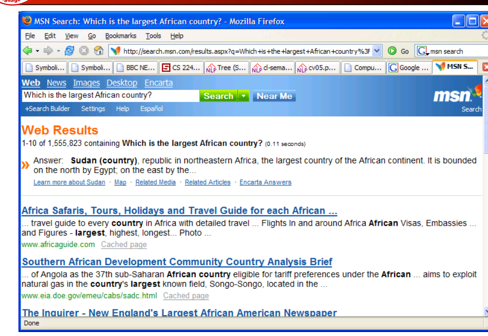
Why study computational semantics?

- 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

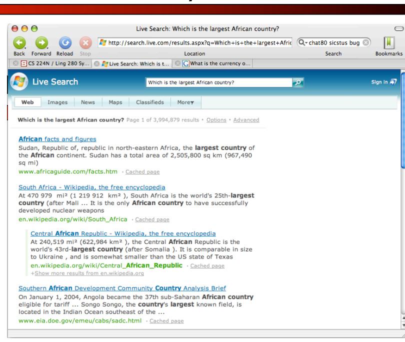
Shallow vs. deep semantics

- 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

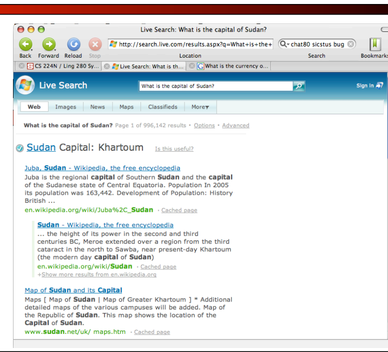
MSN Search: Which is the largest African country?



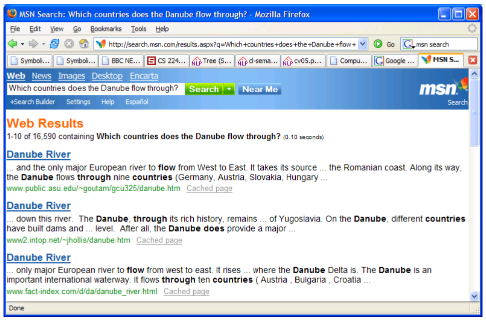
Live Search: Which is the largest African country?



Live Search: What is the capital of Sudan?



MSN Search: Which countries does the Danube flow through?



MSN Search: What are the capitals of the countries bordering the Baltic?

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
  - BUT UNFORTUNATELY PROLOG PORTING ISSUES THIS YEAR ☹

The CHAT-80 Database

```
% Facts about countries.
% country(Country,Region,Latitude,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,?).
country(argentina,south_america,-35,66, 1072067,
   23920000,buenos_aires,peso).

capital(C,Cap) :- country(C,_,_,_,_,_,Cap,_).
```

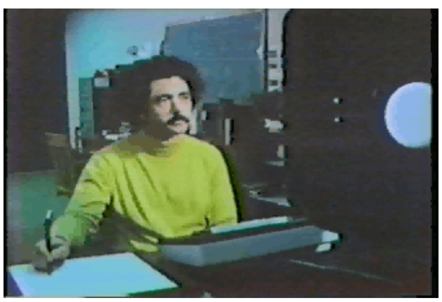
Chat-80 trace (small)

```
Question: What is the capital of
    Australia?
Parse: 0.0sec.
whq
  $VAR
    1
  s
    np
      3+sin
      wh(B)
      []
    verb(be,active,pres+fin,[],pos)
    arg
      dir
      np
        3+sin
        np_head
          det(the(sin))
          []
          capital
        pp
          prep(of)
          np
            3+sin
            name(australia)
            []
    []

Semantics: 0.0sec.
answer([B]) :-
   capital(australia,B)

canberra.
```

Things you could have asked…

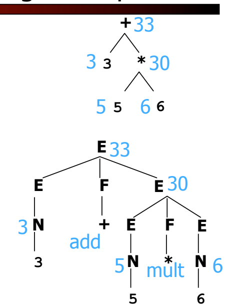
- What is the total area of countries south of the Equator and not in Australasia?
- What is the average area of the countries in each continent?
- Is there more than one country in each continent?
- What are the countries from which a river flows into the Black_Sea?
- Which country bordering the Mediterranean borders a country that is bordered by a country whose population exceeds the population of India?

SHRDLU



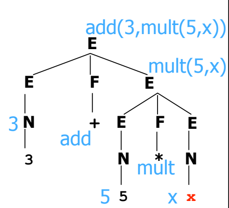
Programming Language Interpreter

- What is meaning of `3+5*6`?
- First parse it into `3+(5*6)`
- Now give a meaning to each node in the tree (bottom-up)



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
- 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 a statement if we can determine its truth
- We understand a 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?? String transduction!
  - English to **logic**? deepest - the definition we'll use!
    - all humans are mortal = $\forall x\ [human(x) \Rightarrow mortal(x)]$
- 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
  - 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 (representing word meaning)
  - Question answering/semantic search/textual entailment

(Formal/Compositional) Computational Semantics

- Sentences: "John smokes."
  "Everyone who smokes snores."
- Syntactic Analyses:

```
      S
    /   \
  NP     VP
  John   smokes
```

- Semantics Construction: smoke(j)
  - Logic as meaning representation language
- Inference: $\forall x.smoke(x) \rightarrow snore(x), smoke(j)$
  => snore(j)

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(x), green(x)
     - 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 square = λp. p*p
  - Equivalent to int square(p) { return p*p; }
  - But we can talk about λp p*p without naming it
  - Format of a lambda term: λ variable . expression

Logic: Lambda Terms

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

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 λx.square(x)
- Similarly, times is equivalent to λx.[λy.times(x,y)]
- Claim that times(5)(6) means same as times(5,6)
  - times(5) = (λx.λy.times(x,y)) (5) = λy.times(5,y)
    - If this function weren't anonymous, what would we call it?
  - times(5)(6) = (λy times(5,y))(6) = times(5,6)
- Referred to as "currying"

Logic: Interesting Constants

- We have "constants" that name some of the entities and functions (e.g., times):
  - GeorgeWBush - 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(GeorgeWBush, LauraBush)
    - *Question:* What does loves(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
  - most(pig, big) = "most pigs are big"
    - Equivalently, most(λx pig(x), λx 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) AND 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 <u>a woman</u> gives birth <u>every 15 min</u>. Our job is to find that woman and stop her.**
  - $\exists$woman ($\forall$15min gives-birth-during(woman, 15min))
  - $\forall$15min ($\exists$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)

In the Background: Model Theory

- $\forall$x.(bird(x) $\rightarrow$ love(s,x)) is a string again!
- Mathematically precise model representation, e.g.: {cat(s), bird(t), love(s,t), granny(g), own(g,s), own(g,t)}
- Inspect formula w.r.t. to the model: Is it true?
- Inferences can extract information: Is anyone not owned by Granny?

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

(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].

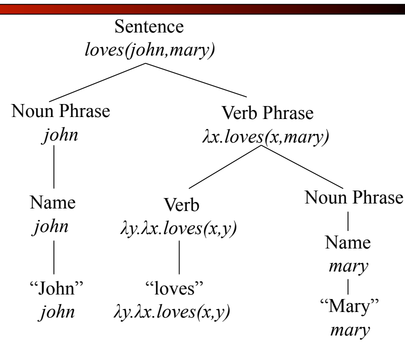
A simple DCG grammar with semantics

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(λx.jumps(x)) --> [jumps]
name(mary) --> [mary].
verb(λy.λx.loves(x,y)) -->[loves]

Combine(X, Y, Z) --> apply(Y, X, Z)

Parse tree with associated semantics



In detail: Beta-Reduction

(λyλx.love(x,y)[mary])[john]
β=> (λx.love(x,mary))[john]
β=> love(john,mary)

Quiz question!

- Suppose the give relation is
  - give(giver, gift, recipient)
- And the order of syntactic composition for
  - *Sue gave Boris a cat*
- is: [Sue [[gave Boris] a cat]
- Write the correct lambda expression for *give*
  - In email, you can write "L" for lambda.

Formal Compositional Semantics …

- **Richard Montague** (1930-1971)
- *"… I reject the contention that an important theoretical difference exists between formal and natural languages …"*

Augmented CFG Rules

- We can also accomplish this just by attaching semantic formation rules to our syntactic CFG rules

$$A \rightarrow \alpha_1 \ldots \alpha_n \quad \{f(\alpha_1.sem, \ldots, \alpha_n.sem)\}$$

- 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

Adding more complex NPs

NP: A man ~> ∃x.man(x)

S: A man loves Mary

~> * love(∃x.man(x),mary)

- How to fix this?

A disappointment

Our first idea for NPs with determiner didn't work out:

"A man" ~> ∃z.man(z)

"A man loves Mary" ~> * love(∃z.man(z),mary)

But what was the idea after all?

Nothing!

∃z.man(z) just isn't the meaning of "a man".

If anything, it translates the complete sentence

"There is a man"

Let's try again, systematically…

A solution

What we want is:

"A man loves Mary" ~> ∃z(man(z) ∧ love(z,mary))

What we have is:

"man" ~> λy.man(y)

"loves Mary" ~> λx.love(x,mary)

How about: ∃z(λy.man(y)(z) ∧ λx.love(x,mary)(z))

Remember: We can use variables for any kind of term.

So next:

λP(λQ.∃z(P(z) ∧Q(z))) <~ "A"

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

An alternative: 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 semantic rules you need
  - 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

  S → <present> the <attribute> of <ship>
  <present> → what is | [can you] tell me
  <attribute> → length | beam | class
  <ship> → the <shipname>
  <shipname> → kennedy | enterprise
  <ship> → <classname> class ships
  <classname> →kitty hawk | lafayette

- Example inputs recognized by above grammar:

  can you tell me the class of the Enterprise
  what is the length of Kitty Hawk class ships

  - 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
    - G. Hendrix, E. Sacerdoti, D. Sagalowicz, and J.Slocum. 1978. Developing a natural language interface to complex data. ACM Transactions on Database Systems 3:105-147

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