Semantics

- **Semantics** = study of meaning
  - We want to investigate the literal meaning of sentences → **compositional semantics**
  - Lexical semantics = study of meaning of words
    - Word Sense Disambiguation deals with lexical semantics
  - To represent the meaning of a sentence, we choose First-Order Predicate Calculus (FOPC) & a basic (Davidsonian) event semantics
    - I have a car
      -  x, y Having (x) ∧ Haver(Speaker,x) ∧ ThingHad(y,x) ∧ Car(y)

Part I: Semantic Representation

There are a variety of way to represent semantics, all sharing some commonalities:

- **Unambiguous** representation: the underlying semantic representation of a sentence should be unambiguous
  - A sentence might mean multiple things
  - But each meaning is represented unambiguously
- Allows for **vagueness**: a semantic representation can be partly undefined
  - I eat Italian food.
  - Not clear exactly what Italian food refers to.
- **Verifiable**: is a particular sentence true or false?

See also Blackburn and Bos (2003), http://www.cogsci.ed.ac.uk/~jbos/comsem/book1.html

Canonical Form

Furthermore, if two distinct sentences mean the same thing, they should have the same semantic representation.

- The **canonical form** is the semantic form for all sentences with the same semantics

  (1) a. Does Maharani have vegetarian dishes?
     b. Do they have vegetarian food at Maharani?
     c. Are vegetarian dishes served at Maharani?
     d. Does Maharani serve vegetarian food?
  - All of these sentences should probably have the same representation (for most purposes)

Model-Theoretic Semantics

Semantic representations are formalized with a **model**

- A model represents the state of affairs in the world being represented
  1. Represent objects, properties of objects, & relations between them
  2. Successfully map the meaning representation to the world being considered

Meaning representation:

- Non-logical vocabulary: names of objects, properties, & relations
  - Denotation: every element of non-logical vocab corresponds to a fixed, well-defined part of model
- Logical vocabulary: closed set of symbols, operators, quantifiers, links, etc.: needed to compose expressions

Denotation

**Extensional** approach to meaning: denotation is reducible to sets

- Domain: set of objects/elements that are part of state of affairs
- Properties: sets of domain elements which have property in question
- Relations: sets of ordered lists/tuples of domain elements

**Interpretation**: Mapping from meaning representations to denotation
Model of restaurant world

- Domain: \[ D = \{ a, b, c, d, e, f, g, h, i, j \} \]
  - Matthew, Franco, Katie, and Caroline: \{a, b, c, d\}
  - Frasca, Med, and Rio: \{e, f, g\}
- Properties
  - Frasca, Med, and Rio are noisy: \text{Noisy} = \{e, f, g\}
- Relations
  - Matthew likes the Med.
  - Katie likes the Med and Rio.
  - Likes = \{<a, f>\} \cup \{<c, f>\} \cup \{<c, g>\}

Predicate-Argument Structure

Recall verb subcategorization requirement

- We can link these syntactic argument slots with semantic roles, or \textit{thematic (theta) roles}

<table>
<thead>
<tr>
<th>Syntactic role</th>
<th>Semantic role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject NP</td>
<td>Agent</td>
</tr>
<tr>
<td>Object NP</td>
<td>Patient</td>
</tr>
</tbody>
</table>

- We can further restrict such theta roles to meet certain conditions, so-called \textit{selectional restrictions}
  - e.g., the agent role of eat must be an animal \[ \rightarrow \]

Towards a Representation

We can represent verbs with semantic roles by:

- defining a semantic predicate for that verb (e.g., \textit{Eat})
- giving the predicate the appropriate number of slots (e.g., 2)

\[ \text{NP}_x \text{ eats NP}_y \Rightarrow \text{Eat}(x, y) \]

- The slots are filled in by \textit{variables} (e.g., \( x, y \)), until we can fill them by actual information from a sentence

Now to define the structures that are allowed ...

First-Order Predicate Calculus (FOPC)

Predicates:

- Predicates take arguments & define the relation among them, e.g. \textit{Eat} takes two arguments (eater/eaten)

Terms, or devices to represent objects:

- \textbf{Constants}: specific objects in the world
e.g., John and fruit in \[ \text{Eat}(\text{John}, \text{fruit}) \]
- \textbf{Variables}: like constants, but not totally specified
e.g., \( x \) in \[ \text{Eat}(\text{John}, x) \Rightarrow \text{no specification of what John eats} \]
- \textbf{Functions}: refer to unique objects which are complex
e.g., the restaurant’s location becomes \[ \text{Loc}(\text{Restaurant}) \]

Logical Connectives

We can build up predicates and then combine them with logical connectives

- \textbf{not (¬)}: I am not happy: \[ \neg \text{Happy} \]
- \textbf{and (\&)}: I am happy and free:
  \[ \text{Happy} \land \text{Free} \]
- \textbf{or (\lor)}: I am happy or I’m free:
  \[ \text{Happy} \lor \text{Free} \]
  - This is an inclusive or: it is true if the speaker is both happy and free (as we’ll see momentarily)
- \textbf{if (⇒)}: If I’m free, then I’m happy:
  \[ \text{Free} \Rightarrow \text{Happy} \]
Variables and Quantifiers

Variables allow a slot to be unfilled, but we need to quantify over (restrict) such variables

- 'there exists' (∃): a restaurant that serves Mexican food: ∃xRestaurant(x) ∧ Serves(x, MexicanFood)
  - Substituting a single restaurant which serves Mexican food for x will make this logical formula true
- 'for all' (∀): All vegetarian restaurants serve vegetarian food: ∀xVegetarianRestaurant(x) ⇒ Serves(x, VegetarianFood)
  - For this to be true, all substitutions for x that make VegetarianRestaurant(x) true must also make Serves(x, VegetarianFood) true

Determining Truth

- Truth-conditional semantics: sentences are analyzed in terms of whether or not they evaluate to true, with respect to some model

To determine whether something is true or not, we evaluate each predicate to see if it's true, and the connectives are interpreted as follows (T=True, F=False):

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>¬p</th>
<th>p ∨ q</th>
<th>p → q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

- Possible-worlds semantics: same idea, but true for a given "possible world"

Rules of Inference

Rules of inference allow us to draw conclusions based on what information we have

- Can add information to database of information

Modus ponens: two statements combine to make a third true:

- All men are mortal (∀x[man(x) → mortal(x)])
- Socrates is a man (man(Socrates))
- Therefore, Socrates is mortal (mortal(Socrates))

Forward/backward chaining

Forward chaining (as in production systems)

- Add individual facts to the knowledge base & use modus ponens to fire implications
- New facts can then cause modus ponens to fire again
- All inference is performed in advance

Backward chaining

- Modus ponens is run in reverse to prove queries
- If query proposition is not in the knowledge base, try to prove it:
  - We don't know if Serves(Leaf, VegetarianFood)
  - But we know: VegetarianRestaurant(Leaf) and VegetarianRestaurant(x) ⇒ Serves(x, VegetarianFood)

Representing Events

A representation like Eats(John, Fruit) and its subsequent meaning postulates can be kind of messy:

- We will instead treat the eating event as a variable:
  - isa(w, Eating) (w is an "isa" Eating event)
  - Actually: there is a w such that this is true: ∃wisa(w, Eating)
- Each argument is then given its own predicate:
  - Eater(w, John), Eaten(w, Fruit)
- Combine them with connectives:
  - ∃wisa(w, Eating) ∧ Eater(w, John) ∧ Eaten(w, Fruit)

This allows us to easily modify these events, e.g., Location(w, RuncibleSpoon)
### Representing Time

New predicates represent time/tense information, to relate sentences to the present moment:

1. I arrive in Peoria
2. a. I arrive in Peoria
   - I believe that Mary ran.
3. a. I arrived in Peoria: ...
   - I believe unicorns exist doesn’t make Unicorns exist true
   - And statements that do mean if
   - If statements that don’t mean if

### Belief: Modal logic

Issues for truth-conditional semantics:

1. I believe that Mary ran.
2. ∀u, vISA(u, Believing) ∧ ISA(v, Running) ∧ Believer(u, Speaker)
   ∧BelievedProp(u, v) ∧ Runner(v, Mary)

What does this actually say?

- There is a running event where Mary was the runner ...
- Verbal aspect: I live in Bloomington vs. I am living in Bloomington
- Belief: I believe unicorns exist doesn’t make Unicorns exist true
- Modal: semantic contribution of may, must, etc.

### Subsumption

To specify hierarchy, we assert subsumption relations

- Restaurant ⊆ CommercialEstablishment
- ItalianRestaurant ⊆ Restaurant
- ChineseRestaurant ⊆ Restaurant

Formally, these are interpreted as subset relations

- Can a restaurant be both Italian and Chinese?
  - Specify disjointness: ChineseRestaurant ⊈ not ItalianRestaurant
  - Fully cover a category: Restaurant ⊆ (or ItalianRestaurant ChineseRestaurant MexicanRestaurant)
Relations

Relations (or roles/role-relations) specify what it means to be a member of a category:
- ItalianCuisine ⊑ Cuisine
- ItalianRestaurant ⊑ Restaurant
- ∃ hasCuisine.ItalianCuisine

Read as: ‘Individuals in the ItalianRestaurant category are subsumed by Restaurant category and an unnamed class: set of entities serving Italian cuisine’
- Existential clause defines unnamed class
- Equivalent FOL: ∀xItalianRestaurant(x) → Restaurant(x) ∧ (∃yServes(x,y) ∧ ItalianCuisine(y))

Inference

Instance checking

Instance checking: determining whether an individual can be classified as a member of a particular category
- Compare known relations & categorical statements to current knowledge
- Return a list of the most specific categories it belongs to

New facts about the individual Gondolier:
- Restaurant(Gondolier)
- hasCuisine(Gondolier, ItalianCuisine)

Can now try to determine if Gondolier is Italian, vegetarian, has moderate prices, etc.

Part II: Deriving a Semantic Analysis

We will focus on two main ways of analyzing the semantics of a sentence:
- Syntax-driven semantic analysis: build up a semantic parse alongside a syntactic parse
  - Requires that we have a semantic form associated with every lexical item and every rule
- Semantic grammars: a more robust way to extract semantic information
  - Not every word will have a semantic form, but we’ll be able to find what we want to find

Augmenting Context-free Rules

Augment context-free rules with semantic attachments

Lexical items (first pass):
- MassNoun → meat {Meat}
- Verb → serves {∃e, x, y Isa(e, Serving) ∧ Server(e, x) ∧ Served(e, y)}

Rules:
- NP → MassNoun {MassNoun.sem}
- VP → Verb NP {Verb.sem(NP.sem)}
Tree Structure

For the phrase serves meat:

\[\text{NP: Meat} \quad \text{MassN: Meat}\]

From existential to instantiated

We would like the semantic value of the VP to be: \[\exists e, x \quad \text{Isa}(e, \text{Serving}) \land \text{Server}(e, x) \land \text{Served}(e, \text{Meat})\]

But how do we go

1. from: \[\exists e, x, y \quad \text{Isa}(e, x) \land \text{Server}(e, x) \land \text{Served}(e, y)\]
2. to: \[\exists e, x \quad \text{Isa}(e, \text{Serving}) \land \text{Server}(e, x) \land \text{Served}(e, \text{Meat})\]

I.e., from “there is a y” to instantiating y as \text{Meat}

Lambdas (Currying)

Instead of saying “there exists a y”, what we want to say is: we have a value of y which is waiting to be filled in.

- A. \lambda (lambda) will do this for us
  - Currying a predicate with multiple arguments into single argument predicates
  - \lambda x P(x) means that x will be replaced by something else, which will then be an argument of P

This is how we apply so-called \lambda-reduction:

- \lambda x P(x)(A)
- P(A)

What about quantifiers?

How do we handle NPs like a restaurant?

- Det \rightarrow a \[\exists\]
- Noun \rightarrow restaurant (Restaurant)
- Nominal \rightarrow Noun \[\lambda x \quad \text{Isa}(x, \text{Noun.sem})\]
- NP \rightarrow Det Nominal (Det.sem x Nominal.sem(x))

The resulting meaning representation will be: \[\exists x \quad \text{Isa}(x, \text{Restaurant})\]

Semantic Problem #1

Quantifier scoping

One major problem we are (for the most part) ignoring is that of quantifier scoping

(8) Every student likes some book

- \forall x \quad [\text{Student}(x) \Rightarrow \exists y \quad [\text{book}(y) \land \text{like}(x, y)]]
- \exists y \quad [\text{book}(y) \land \forall x \quad [\text{Student}(x) \Rightarrow \text{like}(x, y)]]

Some solutions for determining quantifier scope:

- Quantifier storage: store quantifiers in the tree until you need them
- Scope underspecification of scope
- Scope heuristics (left-to-right; domain-specific heuristics; etc.)
Hole semantics

A different approach to underspecifying meaning is that of hole semantics

- \( \lambda \)-variables are replaced with holes
- All FOL subexpressions are given labels
  - dominance constraints restrict which labels can fill which holes
  - e.g., \( l \leq h \): expression containing hole \( h \) dominates expression with label \( l \)

\textit{Every restaurant has a menu:}

\begin{align*}
l_1 & : \forall x \text{Restaurant}(x) \Rightarrow h_1 \\
l_2 & : \exists y \text{Menu}(y) \wedge h_2 \\
l_3 & : \exists e \text{Having}(e) \wedge \text{Haver}(e, x) \wedge \text{Had}(e, y) \\
l_1 & \leq h_0, l_2 \leq h_0, l_3 \leq h_1, l_3 \leq h_2
\end{align*}

Advantages of hole semantics

1. Not dependent upon any particular grammatical construction (e.g., NPs)
   - Can label or designate as holes any arbitrary FOL formula
2. Dominance constraints can rule out unwanted constraints, but without fully specifying the meaning
   - Constraints can come from specific lexical & syntactic knowledge

Semantic Problem #2

Intersecting vs. Scoping Adjectives

Consider the following:

(9) cheap restaurant: \( \lambda x \) \( \text{Isa}(x, \text{Restaurant}) \wedge \text{Isa}(x, \text{Cheap}) \)
(10) a. small elephant \( \rightarrow \) an elephant is not a small thing (only in relation to other elephants)
    b. fake gun \( \rightarrow \) a fake gun is not a gun

- \( \lambda \)-variable is intersective, simply intersecting the semantics of cheap with restaurant
- small elephant is sort of intersective, but small has to be interpreted w.r.t. a context
- fake gun involves an adjective which scopes over the noun, so its semantics should resemble a verb’s: Fake(Gun(x))
Parsing with Semantic Constraints

Can use our semantic information to restrict our parses, e.g., in an Earley parser

(11) # The tree ate my dinner.

Alter the Earley algorithm:
- Keep a field for semantic attachments
- Unify syntactic trees, if able
- Compute semantic analysis and note if it is a valid meaning representation (or perhaps conflicts with what is in the information database)

Semantic Grammars

Instead of mapping semantic rules to syntactic rules, we could just write semantic rules instead.
- Nominal \[\rightarrow\] AdjNominal is split up into rules like FoodType \[\rightarrow\] Nationality FoodType
- This becomes close to template filling: InfoRequest \[\rightarrow\] when does Flight arrive in City

Advantages:
- Previous example will work even with a sentence like When does it arrive in Dallas?
- Avoid dealing with syntactic constituents that have virtually no meaning or add vacuous meaning

Disadvantages of Semantic Grammars

- Not easily reusable ... e.g., have to be talking about flights
- Have a huge explosion of rules
  - vegetarian restaurant, California restaurant, expensive restaurant, and pasta restaurant all need different entries
- Doesn’t match linguistic theory, or intuitions about what happens with language processing

Typically work best in restricted domains