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
- We want a way to represent the meaning of a sentence
  - We'll use First-Order Predicate Calculus (FOPC) and a basic (Davidsonian) event semantics
    - I have a car
    - ∃x, y Having (x) ∧ Haver(Speaker,x) ∧ ThingHad(y,x) ∧ Car(y)

Part I: Semantic Representations

There are a variety of ways to represent semantics, and the best ones depend upon your application, but they share some commonalities:

- **Unambiguous** representation: the underlying semantic representation of a sentence should be unambiguous
  - A sentence might mean multiple things
  - But each one of those meanings should be 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) ... which can be quite difficult to do.

Model-Theoretic Semantics

Semantic representations are formalized with a **model**

- A model represents the state of affairs in the world we're trying to represent
  - represent objects, properties of objects, and relations between them
  - 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, & Caroline: $a, b, c, d$
  - Frasca, Med, Rio: $e, f, g$
- Properties
  - Frasca, Med, and Rio are noisy: $Noisy = \{e, f, g\}$
- Relations
  - Matthew likes the Med.
  - Katie likes the Med and Rio.
  - Likes = $\{<a, f>, <c, f>, <c, g>\}$

Predicate-Argument Structure

Remember how verbs have subcategorization requirements? For example, in John eats Italian food, eats selects for a subject NP and an object NP

- We can link these syntactic argument slots with semantic roles, or 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>

- And we can further restrict such theta roles to meet certain conditions, e.g., the agent role of eat must be an animal → selectional restriction

Towards a Representation

If words like verbs have semantic roles, we can represent that by

- defining a semantic predicate for that verb (e.g. Eat)
- and giving that predicate the appropriate number of slots (e.g., 2)
  \[NP_x \text{ eats } NP_y \Rightarrow \text{Eat}(x, y)\]
- The slots are filled in by variables (e.g., $x, y$), until we can fill them by actual information from a sentence

This idea is relatively straightforward, but we must define what types of structures we allow ...

First-Order Predicate Calculus (FOPC)

Predicates:

- Predicates are essentially verbs: they take arguments and define the relation among them, e.g. Eat takes two arguments (eater and eaten)
- Terms, or devices to represent objects:
  - Constants: specific objects in the world
    - e.g., John and fruit in Eat(John, fruit)
  - Variables: like constants, but undefined, i.e. not totally specified
    - e.g., $x$ in Eat($John, x$) → we haven’t specified what John eats
  - Functions: refer to unique objects which are complex
    - e.g., the restaurant’s location becomes LocationOf(Restaurant)

Why FOPC?

Advantages of first-order predicate calculus (FOPC):

- Proving FOPC statements is efficient
- FOPC statements can be linked to syntactic rules
- FOPC deals with a wide range of linguistic phenomena

Logical Connectives

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

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

Variables give us more power because we can leave a slot unfilled, but we need to **quantify** over such variables; i.e., we need to know what restricts the variables

- ‘there exists’ (∃): a restaurant that serves Mexican food: 
  \[ ∃x \text{Restaurant}(x) ∧ \text{Serves}(x, \text{MexicanFood}) \]
  Substituting a single restaurant which serves Mexican food for \( x \) will make this logical formula true

- ‘for all’ (∀): All vegetarian restaurants serve vegetarian food:
  \[ ∀x \text{VegetarianRestaurant}(x) ⇒ \text{Serves}(x, \text{VegetarianFood}) \]
  For this to be true, all substitutions for \( x \) that make \( \text{VegetarianRestaurant}(x) \) true must also make \( \text{Serves}(x, \text{VegetarianFood}) \) 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>
<th>p → q</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</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

We also have rules of **inference**, which allow us to draw conclusions based on what information we have.

- Allows us to add information to our 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 \( \text{Serves}(\text{Leaf}, \text{VegetarianFood}) \)
    - But we know: \( \text{VegetarianRestaurant}(\text{Leaf}) \) and \( \text{VegetarianRestaurant}(x) ⇒ \text{Serves}(x, \text{VegetarianFood}) \)

### Variables and Quantifiers

What’s wrong with a representation like \( \text{Eats}(\text{John}, \text{Fruit}) \)?

- Is it the same event as \( \text{Eats}_z(\text{John}, \text{Fruit}, \text{Table}) \) (John eats fruit at the table)?
- Could make a meaning postulate:
  \[ ∀x, y, z \text{Eats}(x, y, z) ⇒ \text{MP} \text{Eats}(x, y) \]

This seems unsatisfactory, however ... although, meaning postulates can generally be used to relate, e.g., Eating and Hunger

### Back to language: Meaning Postulates

### Representing Events

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

- We will instead treat the eating event as a variable:
  - \( \text{Isa}(w, \text{Eating}) \) (w is an (“isa”) Eating event)
  - We’ll actually want to say that there is a \( w \) such that this is true:
    \[ ∃w \text{Isa}(w, \text{Eating}) \]
- Each argument is then given its own predicate: \( \text{Eater}(w, \text{John}), \text{Eaten}(w, \text{Fruit}) \)
- We then combine them with connectives:
  \[ ∃w \text{Isa}(w, \text{Eating}) ∧ \text{Eater}(w, \text{John}) ∧ \text{Eaten}(w, \text{Fruit}) \]

This allows us to easily modify these events, e.g., \( \text{Location}(w, \text{TacoBell}) \)
Representing Time

Can add new predicates to represent time/tense information, to understand how the sentences relate to the present moment:

1. I arrive in Peoria
   - \( \exists w, v, t \text{ ISA}(w, \text{Arriving}) \land \text{Arriver}(v, \text{Speaker}) \land \text{Destination}(w, \text{Peoria}) \)

2. I will arrive in Peoria: ...
   - \( \exists w, i \land \text{Interval}(w, i) \land \text{EndPoint}(i, e) \land \text{Precedes}(e, \text{Now}) \)

3. I will will arrive in Peoria: ...
   - \( \exists w, i \land \text{Interval}(w, i) \land \text{MemberOf}(i, \text{Now}) \land \text{Precedes}(\text{Now}, e) \)

More constructions

Likewise, we can augment our semantic representations to handle:

- **Verbal aspect**: I live in Bloomington vs. I am living in Bloomington
- **Belief**: I believe unicorns exist doesn’t make Unicorns exist true
- **Modals**: semantic contribution of may, must, etc.

We’ll look only at belief, and that very quickly.

Belief: Modal logic

Truth-conditional semantics has some issues to deal with:

1. I believe that Mary ran.
2. If you’re interested in baseball, the Rockies are playing tonight.

What does this actually say?

- There is a running event where Mary was the runner ... which may not be true
- There is a running event where Mary was the runner ...

- **Better representation**, using the **modal operator** Believes:
  - Believes(\( \text{Speaker}, \exists v \text{ ISA}(v, \text{Running}) \land \text{Runner}(v, \text{Mary}) \))

- **Modal logics** allow embedding of predicates, at the cost of greater complexity and issues about inferencing, quantification, and so forth

Shortcomings of FOPC by itself

There’s often a difficulty in figuring out what logical connectives are involved

- **if** statements that don’t mean **if**
- **and** statements that do mean **if**

Furthermore, there is the problem that constants like Vegetarian Food have no relation to constants like Vegetarian Restaurant

Description Logics

An alternate representation

Semantic networks: objects are nodes in a graph, and relations are named links between objects

- **Description logics** specify the semantics of structured network representations

Emphasize representation of knowledge about categories, individuals belonging to those categories, & relationships among individuals

- **Terminology**: set of concepts making up a domain
- **TBox**: portion of knowledge base containing terminology
- **ABox**: portion of knowledge base containing facts about individuals
- **Ontology**: captures subset/superset relations among categories

Subsumption

To specify hierarchy, we assert **subsumption** relations

- **Restaurant \( \sqsubseteq \)** Commercial Establishment
- **ItalianRestaurant \( \sqsubseteq \)** Restaurant
- **ChineseRestaurant \( \sqsubseteq \)** Restaurant

Formally, these are interpreted as subset relations

- Can a restaurant be both Italian and Chinese?
  - Specify disjointness: ChineseRestaurant \( \sqsubseteq \) ItalianRestaurant
  - Fully cover a category: Restaurant \( \sqsubseteq \) (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))

25

Inference

Subsumption

Subsumption checks, based on the facts in a terminology, whether a superset/subset relation exists between 2 concepts

Assume that we have defined Italian Restaurants as follows:

- ItalianRestaurant ⊑ Restaurant ⊓∃ hasCuisine. ItalianCuisine
- IlFornaio ⊑ ModerateRestaurant ⊓∃ Cuisine. ItalianCuisine

Subsumption can then check whether the following fact is true:

- IlFornaio ⊑ ItalianRestaurant
  - ModerateRestaurant ⊑ Restaurant
  - ∃ Cuisine. ItalianCuisine restriction is met

26

Inference

Instance checking

Instance checking is the task of 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.

27

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

28

Principle of Compositionality

The meaning of a sentence is composed of the meaning of its parts

- In other words, the way we syntactically compose a sentence determines how we semantically compose it.
- For every syntactic rule, there is a corresponding semantic rule (rule-to-rule hypothesis)

So, a semantic analyzer can take the output of a parser and figure out the semantic meaning

29

Augmenting Context-free Rules

Thus, we can 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)}
What about quantifiers?

How do we handle NPs like a restaurant?

- Det → a {∃}
- Noun → restaurant {Restaurant}
- Nominal → Noun {λx Isa(x, Noun.sem)}
- NP → Det Nominal {Det.sem x Nominal.sem(x)}

The resulting meaning representation will be: ∃x Isa(x, Restaurant)

From existential to instantiated

We would like the semantic value of the VP to be: ∃x, y Isa(e, Serving) ∧ Server(e, x) ∧ Served(e, y)

But how do we go:
- from: ∃x, y Isa(e, Serving) ∧ Server(e, x) ∧ Served(e, y)
- to: ∃x Isa(e, Serving) ∧ Server(e, x) ∧ Served(e, Meat)

We went from saying “there is a y” to instantiating y as Meat, but we have no way to allow this (yet)

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 λ (lambda) will do this for us
  - Currying a predicate with multiple arguments into single argument predicates
- λxP(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 λ-reduction:
- λxP(x)(A)
- P(A)

A revised lexical entry for serves

- Verb → serves {λyλx∃e Isa(e, Serving) ∧ Server(e, x) ∧ Served(e, y)}

- This says: we will first take an argument for y and then put it into the Server relation and then take an argument for x, which goes into the Server relation

And for our rule VP → Verb NP {Verb.sem(NP.sem)}, with NP.sem = Meat, we have the following:
- λy[λx∃e Isa(e, Serving) ∧ Server(e, x) ∧ Served(e, y)](Meat)
- λx∃e Isa(e, Serving) ∧ Server(e, x) ∧ Served(e, Meat)

Tree Structure

For the phrase serves meat:

VP:??

V:∃x, y Isa(e, Serving)...

NP:Meat

MassN:Meat

Semantic Problem #1: Quantifier scoping

Determining the appropriate quantifier scope is a tricky problem. Some solutions:

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

What about quantifiers?

One major problem we are (for the most part) ignoring is that of quantifier scoping.
**Store and Retrieve Approaches**

First, we need *underspecified representations* that embody all readings without enumerating all of them.

**Cooper storage:**

- Replace single semantic attachments with a store
  - Core meaning representation
  - Indexed list of quantified expressions gathered from nodes below this one
  - \( \lambda \)-expressions that combine with core meaning to incorporate quantifiers in the right way

Top node of a parse tree for *Every restaurant has a menu*:

\[ \exists e \text{Having}(e) \land \text{Haver}(e, s_1) \land \text{Havd}(e, s_2) \]
\[ (\lambda Q. \forall x \text{Restaurant}(x) \Rightarrow Q(x), 1), (\lambda Q. \exists x \text{Menu}(x) \land Q(x), 2) \]

---

**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 \( l \)

*Every restaurant has a menu*:

\[ l_1 : \forall x \text{Restaurant}(x) \Rightarrow h_1 \]
\[ l_2 : \exists y \text{Menu}(y) \land h_2 \]
\[ l_3 : \exists e \text{Having}(e) \land \text{Haver}(e, x) \land \text{Havd}(e, y) \]
\[ l_1 \leq h_0, l_2 \leq h_0, l_3 \leq h_1, l_3 \leq h_2 \]

Now, need a *plugging* method to fill the holes

- Can fill \( h_0 \) with either \( l_1 \) or \( l_2 \) as \( h_0 \) dominates both and neither one dominates the other
- e.g., \( P(h_0) = l_1 \), which then leads to \( P(h_1) = l_2 \) and \( P(h_2) = l_3 \)

---

**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: \( \lambda x \text{Isa}(x, \text{Restaurant}) \land \text{Isa}(x, \text{Cheap}) \)
(10) a. small elephant → an elephant is not a small thing (only in relation to other elephants)
    b. fake gun → a fake gun is not a gun

- *cheap restaurant* 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: \( \text{Fake}(\text{Gun}(x)) \)
**Parsing with Semantic Constraints**

Not only can we build up a meaning representation, but we 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