We want to investigate the literal meaning of sentences. Successfully map the meaning representation to the Logical vocabulary: closed set of symbols, operators, quanitifiers, 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.
Frasca, Med, and Rio are noisy:

Matthew likes the Med.
Katie likes the Med and Rio.

Likes = \{<a,f> , <c,f> , <c,g>\}

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

Towards a Representation

We can represent verbs with semantic roles by:
- defining a semantic predicate for that verb (e.g. Eat)
- giving the predicate the appropriate number of slots (e.g., 2)

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

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

Now to define the structures that are allowed ...

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 (\(\wedge\)): I am happy and free:
  \(\text{Happy}(\text{Speaker}) \wedge \text{Free}(\text{Speaker})\)
- 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 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>
<th>p⇒q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</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>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</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)

What’s wrong with a representation like Eats(John, Fruit)?

- Is it the same event as Eats2(John, Fruit, Table)
  - (where John eats fruit at the table)?
- Could make a meaningful postulate:
  - ∃x,y,zEats(x,y,z) ⇒ MP Eats(x,y)

Meaning postulates can generally be used to relate, e.g., Eating and Hunger, but it seems unsatisfactory here
Representing Time

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

(2) a. I arrive in Peoria
   b. ℐ( IntelliJ(w, Arriving) ∧ Arriver(w, Speaker) ∧ Destination(w, Peoria))

(3) a. I arrived in Peoria: ...
   ∧Interval(w, i) ∧ EndPoint(i, e) ∧ Precedes(Now, e)
   b. I am arriving in Peoria: ...
   ∧Interval(w, i) ∧ MemberOf(i, Now)
   c. I will arrive in Peoria: ...
   ∧Interval(w, i) ∧ EndPoint(i, e) ∧ Precedes(Now, e)

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

(4) a. If you’re interested in baseball, the Rockies are playing tonight.
   b. ?? HaveInterestIn(Hearer, Baseball) ⇒ Playing(Rockies, Tonight)

▷ and statements that do mean if

(5) a. One more beer, and I’ll fall off this stool
   b. ?? Beer... ∧ Fall...

Furthermore, constants like VegetarianFood have no relation to constants like VegetarianRestaurant

Description Logics

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 ⊆ CommercialEstablishment
- ItalianRestaurant ⊆ Restaurant
- ChineseRestaurant ⊆ Restaurant

Formally, these are interpreted as subset relations

Can a restaurant be both Italian and Chinese?

- Specify disjointness: ChineseRestaurant ⊆ ¬ 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

Subsumption

Based on the facts in a terminology, subsumption checks if a superset/subset relation exists between 2 concepts.

Assume that we have defined Italian Restaurants as follows:
- ItalianRestaurant ⊑ Restaurant
- hasCuisine(ItalianCuisine)

and we then add this fact:
- IlFornaio ⊑ Restaurant
- hasCuisine(ItalianCuisine)

Subsumption checks whether the following fact is true:
- IlFornaio ⊑ ItalianRestaurant
  - ModerateRestaurant ⊑ Restaurant
  - ∃ Cuisine(ItalianCuisine) restriction is met

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))

Principle of Compositionality

The meaning of a sentence is composed of the meaning of its parts:
- 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)

Semantic analyzer: take the output of a parser & figure out the meaning

Tree Structure

For the phrase serves meat:

V:∃e, x, y Isa(e, Serving)...
NP:Meat
MassN:Meat
Take an argument for Currying

λ

Every student likes some book from:
Nominal
Scope heuristics (left-to-right; domain-specific Quantifier storage: store quantifiers in the tree until you
NP
Semantic underspecification of scope
Verb
λ

It's a predicate with multiple arguments into single argument predicates

This is how we apply so-called λ-reduction:

What about quantifiers?

How do we handle NPs like a restaurant?

1. Take an argument for y & put it into the Served relation
2. Take an argument for x & put it into the Server relation

For VP → Verb NP (Verb.sem(NP.sem)), with NP.sem = Meat, we have the following:

A revised lexical entry for serves

Verb → serves λy.λx.∃e
isa(e, Serving) ∧ Server(e, x) ∧ Served(e, y)

1. Take an argument for y & put it into the Served relation
2. Take an argument for x & put it into the Server relation

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

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)

From existential to instantiated

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

But how do we go

from: ∃e, x, y
isa(e, Serving) ∧ Server(e, x) ∧ Served(e, y)
to: ∃e, x
isa(e, Serving) ∧ Server(e, x) ∧ Served(e, y)
i.e., from “there is a y” to instantiating y as Meat

Some solutions for determining quantifier scope:

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

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
  - λ-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:

∃e Having(e) ∧ Haver(e, s1) ∧ Haved(e, s2)
(λQ.∀xRestaurant(x) ⇒ Q(x), 1).
(λQ.∃xMenu(x) ∧ Q(x), 2)

Semantic Problem #1
Quantifier scoping

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

(6) Every student likes some book

∀x [Student(x) ⇒ ∃y [book(y) ∧ like(x, y)]]

∃y [book(y) ∧ ∀x [Student(x) ⇒ like(x, y)]]

Some solutions for determining quantifier scope:

- Quantifier storage: store quantifiers in the tree until you need them
- Semantic underspecification of scope
- Scope heuristics (left-to-right; domain-specific heuristics; etc.)
Store and Retrieve Approaches (2)

Every restaurant has a menu

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:
\[
\begin{align*}
\lambda_1 : & \forall x \text{Restaurant}(x) \Rightarrow h_1 \\
\lambda_2 : & \exists y \text{Menu}(y) \land h_2 \\
\lambda_3 : & \exists e \text{Having}(e) \land \text{Haver}(e, x) \land \text{Hav}(e, y) \\
\lambda_1 \leq & h_0, \lambda_2 \leq h_0, \lambda_3 \leq h_1, \lambda_3 \leq h_2
\end{align*}
\]

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:

7. cheap restaurant: \( \lambda x \text{Isa}(x, \text{Restaurant}) \land \text{Isa}(x, \text{Cheap}) \)
8. 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

Parsing with Semantic Constraints

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

9. # 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 → AdjNominal** is split up into rules like
  - FoodType → Nationality FoodType
- This becomes close to template filling: InfoRequest → 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