**Semantic Analysis**

- From representation to analysis.
- Syntax-driven semantic analysis.
- From meaning of words to meaning of phrases and sentences.
- Assume given legal parse tree
- Represent meaning of sentence in isolation.

**Finding Meaning**

- Meaning of sentence will be term of FOL
- Meanings of words will be used to build meaning of sentence.
- Parse tree will determine how to combine the meanings.
- Express meanings in terms of what is needed to get a complete sentence.

**Lambda Calculus**

- Convenient way to write “anonymous” functions.
- Two ways to build (untyped) terms:
  - (M N) – function application
  - λx. M – function definition
- Computation rules
  - (α) λx. M = λy. M[y/x] if y not occur in M
  - (β) ((λx. M) N) = M[N/x] if N freely substitutable for x in M

**Typed λ-calculus**

- Specify types of formal parameters:
  - λx:T. M
- Given assignment Γ of types to free variables, can derive types of terms.
- Typed term M is legal w.r.t Γ iff there is a T s.t. Γ ⊢ M: T.
**Typed \(\lambda\)-Calculus**

- Erasures of all terms of typed \(\lambda\)-calculus are terms of untyped \(\lambda\)-calculus, but not vice-versa.
- \( Y = \lambda f. (\lambda x. f(x \ x)) \ (\lambda x. f(x \ x)) \)
- If no constants, terms of typed \(\lambda\)-calculus all converge to a normal form.
  - Halting problems solvable.
- Can’t express recursion without adding extra operators.

**Our \(\lambda\)-Calculus**

- Base type Form: formulas of FOL.
- Could use extensions of FOL as needed.
- Use typed lambda calculus to provide intuition!
- Can add fixed point operators if necessary.

**Finding Meaning**

- Verb phrase needs subject in order to get meaning:
  - \( \text{VPType} = \text{NPType} \rightarrow \text{Form} \)
- Intransitive, transitive, and ditransitive verbs have different meanings:
  - \( \text{IVType} = \text{VType} \)
  - \( \text{TVType} = \text{NPType} \rightarrow \text{VType} \)
  - \( \text{DTVType} = \text{NPType} \rightarrow \text{NPType} \rightarrow \text{VType} \)

**Examples**

- \([[\text{walked}]] = \lambda s: \text{NPType}. \text{walked}(s)\)
- \([[\text{ate}]] = \lambda o: \text{NPType}. \ lambda s: \text{NPType}. \text{ate}(s, o)\)
  - \( \lambda o: \text{NPType}. \lambda s: \text{NPType}. \exists e. \text{Eating}(e) \land \text{Eater}(e, s) \land \text{Eaten}(e, o) \)
- \([[\text{threw}]] = \lambda r: \text{NPType}. \lambda o: \text{NPType}. \lambda s: \text{NPType}. \exists e. \text{Throwing}(e) \land \text{Thrower}(e, s) \land \text{Thrown}(e, o) \land \text{Receiver}(e, r) \)
  - Stick w/simpler non-event representation for now.

**Meaning of Noun Phrases**

- What is NType??
  - Ex: “Jane walked”
  - \([[\text{walked}]] \ [[\text{Jane}]] = \text{walked}([[\text{Jane}]]\)
    - so could let \([[\text{Jane}]] = \text{Jane}, \) a constant.
  - Let NType = D, domain of model

**Not so Fast ...**

- What about \([[\text{All girls walked}]])??
  - \( \forall s. (\text{girl}(s) \Rightarrow \text{walked}(s)) \)
  - “All girls” is noun phrase.
  - Calculate meaning as
    - \((\lambda s: \text{NType}. \text{walked}(s))([[\text{all girls}]])) ?
  - Can’t get meaning that way!
Mathematicians base everything on sets
Computer Scientists on functions
Replace set $S \subseteq D$ by characteristic function $f_S: D \to \text{Form}$
$f_S(x)$ is true in model iff $x \in S$
Binary relation $R$ replaced by $g_R: D \to D$

Form $f_S(x)$ is true in model iff $x(S)$

Binary relation $R$ replaced by $g_R: D \to D$

Form $g_R(d)$

Can represent element $d \in D$ by $f_d: (D \to \text{Form}) \to \text{Form}$
$s.t. f_d(R) = R(d)$

Characterize $d$ extensionally by set of all properties that hold of it.

$\text{NPType} = (D \to \text{Form}) \to \text{Form}$

Note $|(D \to \text{Form}) \to \text{Form}| > |D|$ so lots of room for NP’s.

What is $[\text{all girls}]$?

$\lambda Q: D \to \text{Form}. \forall x. (\text{girl}(x) \Rightarrow Q(x))$

Notice $x$ ranges over els of $D$.

$[\text{all}] = \lambda p: D \to \text{Form}. \lambda Q: D \to \text{Form.}$

$\forall x. (p(x) \Rightarrow Q(x))$

Notice $\text{NounType} = D \to \text{Form}$ and $\lambda Q: D \to \text{Form}$

$\text{DetType} = \text{NounType} \to \text{NounType} \to \text{Form}$

Verb Phrases, Redux

$[[\text{walked}]] = \lambda s: D. \text{walked}(s) \checkmark$

IVerbType = VPType = $D \to \text{Form}$

Transitive verbs:

$[[\text{ate a chicken}]] = \lambda o: \text{NPType}. \lambda s: D. \text{ate}(s, o)$

$[[\text{ate}]] = \lambda o: \text{NPType}. \lambda s: D. o(\text{ate}(s, o))$

so $\lambda s: D. \text{ate}(s, o)$

Transitive Verbs

Computing:

$[[\text{ate a chicken}]] = [[\text{ate}]]([[\text{a chicken}]])$

$(\lambda o: \text{NPType}. \lambda s: D. o(\text{ate}(s, o)))([[\text{a chicken}]])$

$\lambda s: D. [[\text{a chicken}]](\lambda y: D. \text{ate}(s, y))$

$\lambda s: D. \exists y: (\text{chicken}(x) \land Q(x))$

$\lambda y: D. \text{ate}(s, y)$

$\lambda s: D. \exists y: (\text{chicken}(x) \land \text{ate}(s, y))$

$\lambda s: D. \exists y: (\text{chicken}(x) \land \text{ate}(s, y))$

What about ditransitive verbs?

$[[\text{threw}]] = \lambda x: ?. \lambda o: ?. \lambda s: D. \ldots \text{threw}(s, \ldots)$
Types of POS

<table>
<thead>
<tr>
<th>Type</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>NounType</td>
<td>D → Form</td>
</tr>
<tr>
<td>VPType</td>
<td>D → Form</td>
</tr>
<tr>
<td>DetType</td>
<td>NounType → VPType → Form</td>
</tr>
<tr>
<td>NPType</td>
<td>VPType → Form</td>
</tr>
<tr>
<td>PropNounType</td>
<td>NPType</td>
</tr>
<tr>
<td>IVerbType</td>
<td>VPType = D → Form</td>
</tr>
<tr>
<td>TVerbType</td>
<td>NPType → VPType</td>
</tr>
<tr>
<td>DTVerbType</td>
<td>NPType → NPType → VPType</td>
</tr>
</tbody>
</table>

Meanings and Grammar

- Associate meanings w/production rules:
  - $S \rightarrow NP \ VP$ → $[NP.sem(VP.sem)]$
  - $NP \rightarrow Det \ Nom$ → $[Det.sem(Nom.sem)]$
  - $NP \rightarrow PropNoun$ → $[PropNoun.sem]$
  - $Nom \rightarrow Noun$ → $[Noun.sem]$
  - $VP \rightarrow IVerb$ → $[IVerb.sem]$
  - $VP \rightarrow TVerb \ NP$ → $[TVerb.sem(NP.sem)]$

Lexical Semantics

<table>
<thead>
<tr>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda P: NounType. \lambda Q: VPType. \forall x.(P(x) \Rightarrow Q(x))$</td>
</tr>
<tr>
<td>$\lambda P: NounType. \lambda Q: VPType. \exists x.(P(x) \land Q(x))$</td>
</tr>
<tr>
<td>$\lambda x: D. \text{chicken}(x)$</td>
</tr>
<tr>
<td>$\lambda P: VPType. P(jane)$</td>
</tr>
<tr>
<td>$\lambda x: D. \text{walked}(x)$</td>
</tr>
<tr>
<td>$\lambda o: NPType. \lambda s: D. \text{ate}(s, o(y: D. \text{ate}(s, y)))$</td>
</tr>
</tbody>
</table>

Semantic Ambiguity

- Some ambiguities arise at semantic level
  - have same parse trees, but different meanings
- Every student read a book.
  - They each picked their own.
  - Some liked it, while others did not.

“Obvious” Semantics

- Montague [1973]: Rewrite sentence:
  - A book, every student read it.
  - “It” creates a hole to be filled:
    - $[(\text{every student read it})] = \lambda x: D. \exists y.(\text{student}(x) \Rightarrow \text{read}(x, y))$
    - $[(\text{a book})] = \lambda P. \exists y.(\text{book}(y) \land P(y))$ with type $VPType \rightarrow Form$. 

What about other meaning?

- Montague [1973]: Rewrite sentence:
  - A book, every student read it.
  - “It” creates a hole to be filled:
    - $[(\text{every student read it})] = \lambda x: D. \forall x.(\text{student}(x) \Rightarrow \text{read}(x, z))$
    - $[(\text{a book})] = \lambda P. \exists y.(\text{book}(y) \land P(y))$ with type $VPType \rightarrow Form$. 


Putting it Together

- [[A book, every student read it]]
  = (λP.∃y.(book(y) ∧ P(y)))
    (λz:D.∀x.(student(x) ⇒ read(x,z)))
  = ∃y.(book(y) ∧ (λz:D.∀x.(student(x) ⇒
    read(x,z))(y)))
  = ∃y.(book(y) ∧ ∀x.(student(x) ⇒ read(x,y)))
- Seems like a trick!

Any Questions?