Meaning Representation and Semantic Analysis

Ling 571
Deep Processing Techniques for NLP
February 9, 2011
Roadmap

- Meaning representation:
  - Event representations
  - Temporal representation

- Semantic Analysis
  - Compositionality and rule-to-rule
  - Semantic attachments
    - Basic
    - Refinements
  - Quantifier scope
  - Earley Parsing and Semantics
### FOL Syntax Summary

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Formula</code></td>
<td><code>AtomicFormula</code></td>
</tr>
<tr>
<td></td>
<td><code>Formula Connective Formula</code></td>
</tr>
<tr>
<td></td>
<td><code>Quantifier Variable,... Formula</code></td>
</tr>
<tr>
<td></td>
<td><code>¬ Formula</code></td>
</tr>
<tr>
<td></td>
<td><code>(Formula)</code></td>
</tr>
<tr>
<td><code>AtomicFormula</code></td>
<td><code>Predicate(Term,...)</code></td>
</tr>
<tr>
<td><code>Term</code></td>
<td><code>Function(Term,...)</code></td>
</tr>
<tr>
<td></td>
<td><code>Constant</code></td>
</tr>
<tr>
<td></td>
<td><code>Variable</code></td>
</tr>
<tr>
<td><code>Connective</code></td>
<td>`∧</td>
</tr>
<tr>
<td><code>Quantifier</code></td>
<td>`∀</td>
</tr>
<tr>
<td><code>Constant</code></td>
<td>`A</td>
</tr>
<tr>
<td><code>Variable</code></td>
<td>`x</td>
</tr>
<tr>
<td><code>Predicate</code></td>
<td>`Serves</td>
</tr>
<tr>
<td><code>Function</code></td>
<td>`LocationOf</td>
</tr>
</tbody>
</table>
Semantics of FOL

- Model-theoretic approach:
  - FOL terms (objects): denote elements in a domain
  - Atomic formulas are:
    - If properties, sets of domain elements
    - If relations, sets of tuples of elements

- Formulas based on logical operators:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$Q$</td>
<td>$\neg P$</td>
<td>$P \land Q$</td>
<td>$P \lor Q$</td>
<td>$P \Rightarrow Q$</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>
Inference

- Standard AI-type logical inference procedures
  - Modus Ponens
  - Forward-chaining, Backward Chaining
  - Abduction
  - Resolution
  - Etc,..

- We’ll assume we have a prover
Representing Events

- Initially, single predicate with some arguments
- Serves(Maharani, IndianFood)
Representing Events

- Initially, single predicate with some arguments
  - Serves(Maharani, IndianFood)
  - Assume \# ags = \# elements in subcategorization frame
Representing Events

- Initially, single predicate with some arguments
  - Serves(Maharani, IndianFood)
  - Assume \( \# \text{ags} = \# \text{elements in subcategorization frame} \)

- Example:
  - I ate.
  - I ate a turkey sandwich.
  - I ate a turkey sandwich at my desk.
  - I ate at my desk.
  - I ate lunch.
  - I ate a turkey sandwich for lunch.
  - I ate a turkey sandwich for lunch at my desk.
Events

- Issues?
Events

- Issues?
- Arity – how can we deal with different #s of arguments?
Events

- Issues?
  - Arity – how can we deal with different #s of arguments?

- One predicate per frame
  - Eating₁(Speaker)
  - Eating₂(Speaker,TS)
  - Eating₃(Speaker,TS,Desk)
  - Eating₄(Speaker,Desk)
  - Eating₅(Speaker,TS,Lunch)
  - Eating₆(Speaker,TS,Lunch,Desk)
Events (Cont’d)

- Good idea?
Events (Cont’d)

- Good idea?
  - Despite the names, actually unrelated predicates
Events (Cont’d)

- Good idea?
  - Despite the names, actually unrelated predicates
    - Can’t derive obvious info
      - E.g. I ate a turkey sandwich for lunch at my desk
        - Entails all other sentences
Events (Cont’d)

- Good idea?
  - Despite the names, actually unrelated predicates
    - Can’t derive obvious info
      - E.g. I ate a turkey sandwich for lunch at my desk
        - Entails all other sentences
    - Can’t directly associate with other predicates
Events (Cont’d)

- Good idea?
  - Despite the names, actually unrelated predicates
    - Can’t derive obvious info
      - E.g. I ate a turkey sandwich for lunch at my desk
        - Entails all other sentences
        - Can’t directly associate with other predicates
  - Could write rules to implement implications
Events (Cont’d)

- Good idea?
  - Despite the names, actually unrelated predicates
    - Can’t derive obvious info
      - E.g. I ate a turkey sandwich for lunch at my desk
        - Entails all other sentences
      - Can’t directly associate with other predicates
  - Could write rules to implement implications
    - But?
      - Intractable in the large
        - Like the subcat problem generally.
Variabilizing

- Create predicate with maximum possible arguments
- Include appropriate args
- Maintains connections
  \[ \exists w, x, y \text{Eating}(\text{Speaker}, w, x, y) \]
  \[ \exists w, x \text{Eating}(\text{Speaker}, \text{TS}, w, x) \]
  \[ \exists w \text{Eating}(\text{Speaker}, \text{TS}, w, \text{Desk}) \]
  \[ \text{Eating}(\text{Speaker}, \text{TS}, \text{Lunch}, \text{Desk}) \]
Variabilizing

- Create predicate with maximum possible arguments
  - Include appropriate args
  - Maintains connections
    \[ \exists w, x, y \text{Eating}(\text{Speaker}, w, x, y) \]
    \[ \exists w, x \text{Eating}(\text{Speaker}, \text{TS}, w, x) \]
    \[ \exists w \text{Eating}(\text{Speaker}, \text{TS}, w, \text{Desk}) \]
    \[ \text{Eating}(\text{Speaker}, \text{TS}, \text{Lunch}, \text{Desk}) \]

- Better?
Variabilizing

- Create predicate with maximum possible arguments
  - Include appropriate args
  - Maintains connections
    \[ \exists w, x, y Eating(Speaker, w, x, y) \]
    \[ \exists w, x Eating(Speaker, TS, w, x) \]
    \[ \exists w Eating(Speaker, TS, w, Desk) \]
    \[ Eating(Speaker, TS, Lunch, Desk) \]

- Better?
  - Yes, but
    - Too many commitments – assume all details show up
Variabilizing

- Create predicate with maximum possible arguments
  - Include appropriate args
  - Maintains connections

\[ \exists w, x, yEating(Speaker, w, x, y) \]
\[ \exists w, xEating(Speaker, TS, w, x) \]
\[ \exists wEating(Speaker, TS, w, Desk) \]
\[ Eating(Speaker, TS, Lunch, Desk) \]

- Better?
  - Yes, but
    - Too many commitments – assume all details show up
    - Can’t individuate – don’t know if same event
Events - Finalized

- Neo-Davidsonian representation:
  - Distill event to single argument for event itself
  - Everything else is additional predication

$\exists e \text{Eating}(e) \land \text{Eater}(e, \text{Speaker}) \land \text{Eaten}(e, \text{TS}) \land \text{Meal}(e, \text{Lunch}) \land \text{Location}(e, \text{Desk})$
Events - Finalized

- Neo-Davidsonian representation:
  - Distill event to single argument for event itself
  - Everything else is additional predication

\[ \exists e \text{Eating}(e) \land \text{Eater}(e, \text{Speaker}) \land \text{Eaten}(e, \text{TS}) \land \text{Meal}(e, \text{Lunch}) \land \text{Location}(e, \text{Desk}) \]

- Pros:
Events - Finalized

- Neo-Davidsonian representation:
  - Distill event to single argument for event itself
  - Everything else is additional predication

\[ \exists e \text{Eating}(e) \land \text{Eater}(e, \text{Speaker}) \land \text{Eaten}(e, TS) \land \text{Meal}(e, \text{Lunch}) \land \text{Location}(e, \text{Desk}) \]

- Pros:
  - No fixed argument structure
    - Dynamically add predicates as necessary
Events - Finalized

- Neo-Davidsonian representation:
  - Distill event to single argument for event itself
  - Everything else is additional predication

\[ \exists e \text{Eating}(e) \land \text{Eater}(e, \text{Speaker}) \land \text{Eaten}(e, \text{TS}) \land \text{Meal}(e, \text{Lunch}) \land \text{Location}(e, \text{Desk}) \]

- Pros:
  - No fixed argument structure
    - Dynamically add predicates as necessary
  - No extra roles
Events - Finalized

- Neo-Davidsonian representation:
  - Distill event to single argument for event itself
  - Everything else is additional predication

\[
\exists e \text{Eating}(e) \land \text{Eater}(e, \text{Speaker}) \land \text{Eaten}(e,TS) \land \text{Meal}(e,\text{Lunch}) \land \text{Location}(e,\text{Desk})
\]

- Pros:
  - No fixed argument structure
    - Dynamically add predicates as necessary
  - No extra roles
  - Logical connections can be derived
Representing Time

- Temporal logic:
  - Includes tense logic to capture verb tense information

- Basic notion:
  - Timeline:
    - From past to future
    - Events associated with points or intervals on line
      - Ordered by positioning on line
    - Current time
      - Relative order gives past/present/future
Temporal Information

- I arrived in New York.
- I am arriving in New York.
- I will arrive in New York.
  - Same event, differ only in tense

\[ \exists e \text{Arriving}(e) \land \text{Arriver}(e, \text{Speaker}) \land \text{Destination}(e, \text{NY}) \]

- Create temporal representation based on verb tense
- Add predication about event variable
- Temporal variables represent:
  - Interval of event
  - End point of event
  - Predicates link end point to current time
Temporal Representation

\[ \exists e, i, n, tArriving(e) \land Arriver(e, \text{Speaker}) \land Destination(e, NY) \land IntervalOf(e, i) \land EndPoint(i, e) \land Precedes(e, Now) \]

\[ \exists e, i, n, tArriving(e) \land Arriver(e, \text{Speaker}) \land Destination(e, NY) \land IntervalOf(e, i) \land MemberOf(i, Now) \]

\[ \exists e, i, n, tArriving(e) \land Arriver(e, \text{Speaker}) \land Destination(e, NY) \land IntervalOf(e, i) \land EndPoint(i, n) \land Precedes(Now, e) \]
More Temp Rep

- Flight 902 arrived late.
- Flight 902 had arrived late.

- Does the current model cover this?
  - Not really
  - Need additional notion:
    - Reference point
    - As well as current time, event time
      - Current model: current = utterance time = reference point
Reichenbach’s Tense Model

Past Perfect
I had eaten

Simple Past
I ate

Present Perfect
I have eaten

Present
I eat

Simple Future
I will eat

Future Perfect
I will have eaten

E, R, U

R, E, U

E, R, U

U, R, E

U, R, E

U, E, R
Meaning Representation for Computational Semantics

- **Requirements:**
  - Verifiability, Unambiguous representation, Canonical Form, Inference, Variables, Expressiveness

- **Solution:**
  - First-Order Logic
    - Structure
    - Semantics
    - Event Representation

- **Next: Semantic Analysis**
  - Deriving a meaning representation for an input
Syntax-driven Semantic Analysis

- Key: Principle of Compositionality
  - Meaning of sentence from meanings of parts
    - E.g. groupings and relations from syntax

- Question: Integration?

- Solution 1: Pipeline
  - Feed parse tree and sentence to semantic unit
  - Sub-Q: Ambiguity:
    - Approach: Keep all analyses, later stages will select
Simple Example

- AyCaramba serves meat.

\[ \exists e \text{ Isa}(e, \text{Serving}) \land \text{Server}(e, \text{AyCaramba}) \land \text{Served}(e, \text{Meat}) \]
Rule-to-Rule

- **Issue:**
  - How do we know which pieces of the semantics link to what part of the analysis?
  - Need detailed information about sentence, parse tree
    - Infinitely many sentences & parse trees
    - Semantic mapping function per parse tree => intractable

- **Solution:**
  - Tie semantics to finite components of grammar
    - E.g. rules & lexicon
  - Augment grammar rules with semantic info
    - Aka “attachments”
    - Specify how RHS elements compose to LHS
Semantic Attachments

• Basic structure:
  • A -> a1....an {f(aj.sem,...ak.sem)}
  • A.sem

• Language for semantic attachments
  • Arbitrary programming language fragments?
    • Arbitrary power but hard to map to logical form
    • No obvious relation between syntactic, semantic elements
  • Lambda calculus
    • Extends First Order Predicate Calculus (FOPC) with function application
    • Feature-based model + unification

• Focus on lambda calculus approach
Basic example

- Input: Maharani closed.
- Target output: Closed(Maharani)
Semantic Analysis Example

- Semantic attachments:
  - Each CFG production gets semantic attachment

- Maharani
  - ProperNoun -> Maharani  {Maharani}
    - FOL constant to refer to object

- NP -> ProperNoun  {ProperNoun.sem}
  - No additional semantic info added
Semantic Attachment

Example

- Phrase semantics is function of SA of children

- More complex functions are parameterized
  - E.g. Verb -> closed \( \{ \lambda x.\text{Closed}(x) \} \)
    - Unary predicate:
      - 1 arg = subject, not yet specified

- VP -> Verb \( \{ \text{Verb.sem} \} \)
  - No added information

- S -> NP VP \( \{ \text{VP.sem(NP.sem)} \} \)
  - Application= \( \lambda x.\text{Closed}(x)(\text{Maharanii}) = \text{Closed}(\text{Maharani}) \)
Semantic Attachment

- General pattern:
  - Grammar rules mostly lambda reductions
    - Functor and arguments
  - Most representation resides in lexicon
Refining Representation

- Add
  - Neo-Davidsonian event-style model
  - Complex quantification

- Example II
  - Input: Every restaurant closed.
  - Target:

\[
\forall x \text{Restaurant}(x) \Rightarrow \exists e \text{Closed}(e) \wedge \text{ClosedThing}(e, x)
\]
Refining Representation

- Idea: $\forall x \text{Restaurant}(x)$
  - Good enough?
    - No: roughly ‘everything is a restaurant’
    - Saying something about all restaurants – nuclear scope

- Solution: Dummy predicate
  $\forall x \text{Restaurant}(x) \Rightarrow Q(x)$
  - Good enough?
    - No: no way to get $Q(x)$ from elsewhere in sentence

- Solution: Lambda
  $\lambda Q. \forall x \text{Restaurant}(x) \Rightarrow Q(x)$
Updating Attachments

- Noun -> restaurant \{ \lambda x. \text{Restaurant}(x) \}\n
- Nominal -> Noun \{ \text{Noun.sem} \}\n
- Det -> Every \{ \lambda P. \lambda Q. \forall x P(x) \Rightarrow Q(x) \}\n
- NP -> Det Nominal \{ \text{Det.sem(Nom.sem)} \}
\[\lambda P. \lambda Q. \forall x P(x) \Rightarrow Q(x)(\lambda x. \text{Restaurant}(x))\]
\[\lambda P. \lambda Q. \forall x P(x) \Rightarrow Q(x)(\lambda y. \text{Restaurant}(y))\]
\[\lambda Q. \forall x \lambda y. \text{Restaurant}(y)(x) \Rightarrow Q(x)\]
\[\lambda Q. \forall x \text{Restaurant}(x) \Rightarrow Q(x)\]
Full Representation

- Verb -> close \{ \lambda x. \exists e \text{Closed}(e) \land \text{ClosedThing}(e, x) \}\}
- VP -> Verb \{ \text{Verb.sem} \}\}
- S -> NP VP \{ \text{NP.sem(VP.sem)} \} \}

\lambda Q. \forall x \text{Restaurant}(x) \Rightarrow Q(x)(\lambda y. \exists e \text{Closed}(e) \land \text{ClosedThing}(e, y))

\forall x \text{Restaurant}(x) \Rightarrow \lambda y. \exists e \text{Closed}(e) \land \text{ClosedThing}(e, y)(x)

\forall x \text{Restaurant}(x) \Rightarrow \exists e \text{Closed}(e) \land \text{ClosedThing}(e, x)
Generalizing Attachments

- ProperNoun -> Maharani {Maharani}

- Does this work in the new style?
  - No, we turned the NP/VP application around

- New style: $\lambda x.x(Maharani)$
More

- Determiner
- Det -> a 
  \[ \lambda P. \lambda Q. \exists x P(x) \land Q(x) \]
- a restaurant
  \[ \lambda Q. \exists x \text{Restaurant}(x) \land Q(x) \]

- Transitive verb:
  - VP -> Verb  NP { Verb.sem(NP.sem) }
  - Verb -> opened
  \[ \lambda w. \lambda z. w(\lambda x. \exists e \text{Opened}(e) \land \text{Opener}(e, z) \land \text{OpenedThing}(e, w) \]
Strategy for Semantic Attachments

- General approach:
  - Create complex, lambda expressions with lexical items
    - Introduce quantifiers, predicates, terms
  - Percolate up semantics from child if non-branching
  - Apply semantics of one child to other through lambda
    - Combine elements, but don’t introduce new
### Sample Attachments

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>Semantic Attachment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>${NP.sem(\text{VP}.sem)}$</td>
</tr>
<tr>
<td>$NP \rightarrow Det \ Nominal$</td>
<td>${Det.sem(\text{Nominal}.sem)}$</td>
</tr>
<tr>
<td>$NP \rightarrow ProperNoun$</td>
<td>${ProperNoun.sem}$</td>
</tr>
<tr>
<td>$Nominal \rightarrow Noun$</td>
<td>${Noun.sem}$</td>
</tr>
<tr>
<td>$VP \rightarrow Verb$</td>
<td>${Verb.sem}$</td>
</tr>
<tr>
<td>$VP \rightarrow Verb \ NP$</td>
<td>${Verb.sem(\text{NP}.sem)}$</td>
</tr>
<tr>
<td>$Det \rightarrow every$</td>
<td>${\lambda P. \lambda Q. \forall x P(x) \Rightarrow Q(x)}$</td>
</tr>
<tr>
<td>$Det \rightarrow a$</td>
<td>${\lambda P. \lambda Q. \exists x P(x) \land Q(x)}$</td>
</tr>
<tr>
<td>$Noun \rightarrow restaurant$</td>
<td>${\lambda r. \text{Restaurant}(r)}$</td>
</tr>
<tr>
<td>$ProperNoun \rightarrow Matthew$</td>
<td>${\lambda m. m(\text{Matthew})}$</td>
</tr>
<tr>
<td>$ProperNoun \rightarrow Franco$</td>
<td>${\lambda f. f(\text{Franco})}$</td>
</tr>
<tr>
<td>$ProperNoun \rightarrow Frasca$</td>
<td>${\lambda f. f(\text{Frasca})}$</td>
</tr>
<tr>
<td>$Verb \rightarrow closed$</td>
<td>${\lambda x. \exists e \text{Closed}(e) \land \text{ClosedThing}(e, x)}$</td>
</tr>
<tr>
<td>$Verb \rightarrow opened$</td>
<td>${\lambda w. \lambda z. w(\lambda x. \exists e \text{Opened}(e) \land \text{Opener}(e, z) \land \text{Opened}(e, x))}$</td>
</tr>
</tbody>
</table>
Quantifier Scope

- Ambiguity:
  - Every restaurant has a menu
  \[ \forall x \text{Restaurant}(x) \Rightarrow \exists y \text{Menu}(y) \land (\exists e \text{Having}(e) \land \text{Haver}(e, x) \land \text{Had}(e, y)) \]
  - Readings:
    - all have a menu;
    - all have same menu
  - Only derived one
  \[ \exists y \text{Menu}(y) \land \forall x (\text{Restaurant}(x) \Rightarrow \exists e \text{Having}(e) \land \text{Haver}(e, x) \land \text{Had}(e, y)) \]
  - Potentially O(n!) scopings (n=\# quantifiers)

- There are approaches to describe ambiguity efficiently and recover all alternatives.
Earley Parsing with Semantics

- Implement semantic analysis
  - In parallel with syntactic parsing
    - Enabled by compositional approach

- Required modifications
  - Augment grammar rules with semantic field
  - Augment chart states with meaning expression
  - Completer computes semantics – e.g. unifies
    - Can also fail to unify
      - Blocks semantically invalid parses
    - Can impose extra work
Sidelight: Idioms

- Not purely compositional
  - E.g. kick the bucket = die
  - tip of the iceberg = beginning

- Handling:
  - Mix lexical items with constituents (word nps)
  - Create idiom-specific const. for productivity
  - Allow non-compositional semantic attachments

- Extremely complex: e.g. metaphor