# Computational Semantics

Deep Processing for NLP
Ling 571
February 8, 2017

# Roadmap

- Compositional Semantics
  - Rule-to-rule model
  - Semantic attachments
  - Extended examples
  - Scope and Parsing

# Summary

- First-order logic can be used as a meaning representation language for natural language
- Principle of compositionality: the meaning of a complex expression is a function of the meaning of its parts
- $\lambda$-expressions can be used to compute meaning representations from syntactic trees based on the principle of compositionality
- In the next section, we will look at a syntax-driven approach to semantic analysis in more detail

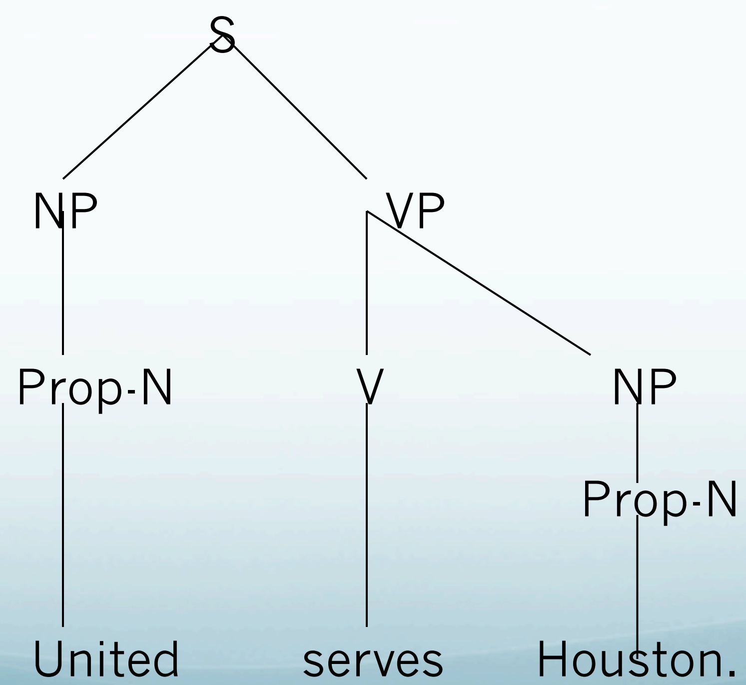
# Syntax-driven Semantic Analysis

- 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

- United serves Houston.

$$\exists e\ Serving(e) \wedge Server(e, United) \wedge Served(e, Houston)$$



# 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→ $a_1$....$a_n$ {f($a_j$.sem,...$a_k$.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

# Semantic Analysis Approach

- Semantic attachments:
  - Each CFG production gets semantic attachment

- Phrase semantics is function of SA of children
  - Complex functions parametrized

    - E.g. Verb → arrived
      - Need unary predicate
        - One arg: subject, not yet available

# Semantic Analysis Example

- Basic model:
  - Neo-Davidsonian event-style model
  - Complex quantification
- Example:
  - Every flight arrived.
  - (S (NP (Det every) (Nom (Noun flight)))
  - (VP (V arrived)))
  - Target representation:

$$\forall x Flight(x) \Rightarrow \exists e Arrived(e) \wedge ArrivedThing(e,x)$$

# Defining Representation

- Idea: Every flight = $\forall x Flight(x)$
  - Good enough?
    - No: roughly 'everything is a flight'
    - Saying something about all flights – nuclear scope
- Solution: Dummy predicate

  $$\forall x Flight(x) \Rightarrow Q(x)$$

  - Good enough?
    - No: no way to get Q(x) from elsewhere in sentence
- Solution: Lambda

  $$\lambda Q.\forall x Flight(x) \Rightarrow Q(x)$$

# Creating Attachments

- Noun → flight $\{\lambda x.Flight(x)\}$
- Nom → Noun { Noun.sem }
- Det → Every $\{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$
- NP → Det Nom { Det.sem(Nom.sem) }

$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda x.Flight(x))$$

$$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)(\lambda y.Flight(y))$$

$$\lambda Q.\forall x \lambda y.Flight(y)(x) \Rightarrow Q(x)$$

$$\lambda Q.\forall x Flight(x) \Rightarrow Q(x)$$

# Full Representation

- Verb → arrived $\{\lambda x.\exists e Arrived(e) \wedge ArrivedThing(e,x)\}$
- VP → Verb { Verb.sem }
- S → NP VP { NP.sem(VP.sem) }

$$\lambda Q.\forall x Flight(x) \Rightarrow Q(x)(\lambda y.\exists e Arrived(e) \wedge ArrivedThing(e,y))$$

$$\forall x Flight(x) \Rightarrow \lambda y.\exists e Arrived(e) \wedge ArrivedThing(e,y)(x)$$

$$\forall x Flight(x) \Rightarrow \exists e Arrived(e) \wedge ArrivedThing(e,x)$$

# Extending Attachments

- ProperNoun → UA223

- What should semantics look like in this style?
  - Needs to produce correct form when applied to VP.sem
    - As in "UA223 arrived" →

$$\exists e Arrived(e) \wedge ArrivedThing(e, UA223)$$

- Correct form: $\lambda$ X.X (UA223)
  - Applies predicate to UA223

# More

- Determiner
- Det → a { $\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)$ }
- a flight $\lambda Q.\exists x Flight(x) \wedge Q(x)$

- Transitive verb:
  - VP → Verb NP { Verb.sem(NP.sem) }
  - Verb → booked

$$\lambda w.\lambda z.w(\lambda x.\exists e Booked(e) \wedge Booker(e,z) \wedge BookedThing(e,x))$$

# 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

# John booked a flight

- a flight $\lambda Q.\exists x Flight(x) \wedge Q(x)$
- VP → Verb NP {Verb.sem(NP.sem)}

$$\lambda w.\lambda z.w(\lambda x.\exists e Booked(e) \wedge Booker(e,z) \wedge BookedThing(e,x))$$

- ( $\lambda Q.\exists y Flight(y) \wedge Q(y)$ )

$$\lambda z.\lambda Q.\exists y Flight(y) \wedge Q(y)$$

$$(\lambda x.\exists e Booked(e) \wedge Booker(e,z) \wedge BookedThing(e,x))$$

$$\lambda z.\exists y Flight(y) \wedge$$

$$\lambda x.\exists e Booked(e) \wedge Booker(e,z) \wedge BookedThing(e,x)(y)$$

# John booked a flight

- a flight $\lambda Q.\exists x Flight(x) \wedge Q(x)$
- VP → Verb NP {Verb.sem(NP.sem)}

$$\lambda w.\lambda z.w(\lambda x.\exists e Booked(e) \wedge Booker(e,z) \wedge BookedThing(e,x))$$

- ( $\lambda Q.\exists y Flight(y) \wedge Q(y)$ )

$$\lambda z.\lambda Q.\exists y Flight(y) \wedge Q(y)$$

$$(\lambda x.\exists e Booked(e) \wedge Booker(e,z) \wedge BookedThing(e,x))$$

$$\lambda z.\exists y Flight(y) \wedge$$

$$\exists e Booked(e) \wedge Booker(e,z) \wedge BookedThing(e,y)$$

# John booked a flight

- Proper_Noun → John {$\lambda$ x.x(John)}
- S → NP VP {NP.sem(VP.sem)}
- $\lambda$ x.x(John)

$$(\lambda z.\exists y Flight(y) \wedge$$
$$\exists e Booked(e) \wedge Booker(e,z) \wedge BookedThing(e,y))$$

$$(\lambda z.\exists y Flight(y) \wedge$$
$$\exists e Booked(e) \wedge Booker(e,z) \wedge BookedThing(e,y))(John)$$

# John booked a flight

$$(\lambda z.\exists y Flight(y) \wedge$$
$$\exists e Booked(e) \wedge Booker(e,z) \wedge BookedThing(e,y))(John)$$

$$\exists y Flight(y) \wedge$$
$$\exists e Booked(e) \wedge Booker(e,John) \wedge BookedThing(e,y)$$

# 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

# Semantics Learning

- Zettlemoyer & Collins, 2005, 2007, etc; Mooney 2007
- Given semantic representation and corpus of parsed sentences
  - Learn mapping from sentences to logical form
    - Structured perceptron
    - Applied to ATIS corpus sentences
- Similar approaches to: learning instructions from computer manuals, game play from walkthroughs, robocup/soccer play from commentary

# Quantifier Scope

- Ambiguity:
  - *Every restaurant has a menu*

$$\forall x \, Restaurant(x) \Rightarrow \exists y (Menu(y) \wedge (\exists e Having(e) \wedge Haver(e,x) \wedge Had(e,y)))$$

- Readings:
  - all have a menu;
  - all have same menu
- Only derived one

$$\exists y Menu(y) \wedge \forall x (Restaurant(x) \Rightarrow \exists e Having(e) \wedge Haver(e,x) \wedge Had(e,y)))$$

- Potentially O(n!) scopings (n=# quantifiers)

- There are approaches to describe ambiguity efficiently and recover all alternatives.

# 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
  - Incrementally compute semantics
    - Can also fail
      - 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