CS 181: Natural Language Processing

Lecture 16: Computational Semantics

Kim Bruce
Pomona College
Spring 2008

Disclaimer: Slide contents borrowed from many sources on web!

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.

Finding Meaning

- Verb phrase needs subject in order to get meaning:
  - VPType = NPType → Form
- Intransitive, transitive, and ditransitive verbs have different meanings:
  - IVType = VPType
  - TVType = NPType → VPType
  - DTVType = NPType → NPType → VPType

Lambda Calculus

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

Typed λ-calculus

- Specify types of formal parameters:
  - $\lambda x{:}T.\ M$
  - Given assignment Γ of types to free variables, can derive types of terms.
  - $\Gamma \cup \{x{:}T\} \vdash M: U$ implies $\Gamma \vdash \lambda x{:}T.\ M: T \to U$
  - $\Gamma \vdash M: T \to U$, $\Gamma \vdash N: T$ implies $\Gamma \vdash (M\ N): U$
- Typed term M is legal w.r.t Γ iff there is a T s.t. $\Gamma \vdash M: T$.

Typed λ-Calculus

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

Our λ-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:
  - VPType = NPType → Form
- Intransitive, transitive, and ditransitive verbs have different meanings:
  - IVType = VPType
  - TVType = NPType → VPType
  - DTVType = NPType → NPType → VPType

Examples

- [[walked]] = λs:NPType. *walked*(s)
- [[ate]] = λo: NPType. λs:NPType. *ate*(s,o)

  or

  λo: NPType. λs:NPType. ∃e. *Eating*(e) ∧ *Eater*(e,s) ∧ *Eaten*(e,o)
- [[threw]] = λr: NPType. λo: NPType. λs:NPType. ∃e. *Throwing*(e) ∧ *Thrower*(e,s) ∧ *Thrown*(e,o) ∧ *Receiver*(e,r)

Stick w/simpler non-event representation for now.

Meaning of Noun Phrases

- What is NPType?
  - Ex: "Jane walked"
  - ([[walked]] [[Jane]]) = *walked*([[Jane]]) so could let [[Jane]] = *Jane*, a constant.
  - Let NPType = D, domain of model

Not so Fast ...

- What about [[All girls walked]]?
  - ∀x.(*girl*(x) ⇒ *walked*(x))
- "All girls" is noun phrase.
- Calculate meaning as
  - (λs:NPType. *walked*(s))([[all girls]]) ?
- Can't get meaning that way!

Back Up

- Mathematicians base everything on sets
- Computer Scientists on functions
- Replace set $S \subseteq D$ by characteristic function f_S: D → Form
  - $f_S(x)$ is true in model iff $x \in S$
- Binary relation R replaced by g_R: D × D → Form

λ-Lifting

- Can represent element $d \in D$ by f_d: (D → Form) → Form s.t.: $f_d(R) = R(d)$.
- Characterize d extensionally by set of all properties that hold of it.
- NPType = (D → Form) → Form
- Note |(D → Form) → Form | >> |D| so lots of room for NP's.

Determiners

- What is [[all girls]]?
  - λQ: D → Form. ∀x. (*girl*(x) ⇒ Q(x))
  - *Notice x ranges over elts of D.*
- [[all]] = λP: D → Form. λQ: D → Form. ∀x. (P(x) ⇒ Q(x))
- [[exists]] = λP: D → Form. λQ: D → Form. ∃x. (P(x) ∧ Q(x))
- Notice NounType = D → Form *and* DetType = NounType → NounType → Form (?)

Back to Verb Phrases

- S → NP VP
- Compute [[S]] = [[NP]]([[VP]]): Form
- NPType = (D → Form) → Form
- Thus VPType = D → Form, not NPType → Form and NPType = VPType → Form
- Fix previous semantics
  - DetType = NounType → VPType → Form

Verb Phrases, Redux

- [[walked]] = λs:D. *walked*(s) ✔
  - IVerbType = VPType = D → Form
- Transitive verbs:
  - [[ate a chicken]] = [[ate]]([[a chicken]])
  - where [[a chicken]] = λQ: D→Form. ∃x(*chicken*(x) ∧ Q(x))
  - [[ate]] = λo: NPType. λs:D. *ate*(s,o) ✘
  - [[ate]] = λo: NPType. λs:D. o(λy: D. *ate*(s,y)) ✔
  - so TVerbType = NPType→ VPType

Transitive Verbs

- Computing:
  - [[ate a chicken]] = [[ate]]([[a chicken]])
    = (λo: NPType. λs:D. o(λy: D. *ate*(s,y))) ([[a chicken]])
    = λs:D. [[a chicken]](λy: D. *ate*(s,y))
    = λs:D. (λQ: D→Form. ∃x(*chicken*(x) ∧ Q(x))) (λy: D. *ate*(s,y))
    = λs:D. ∃x(*chicken*(x) ∧ (λy: D. *ate*(s,y))(x))
    = λs:D. ∃x(*chicken*(x) ∧ *ate*(s,x))
- What about ditransitive verbs?
  - [[threw]] = λr: ?. λo: ?. λs:D. …*Threw*(s,…,..)…

Types of POS

NounType	D → Form
VPType	D → Form
DetType	NounType → VPType → Form
NPType	VPType → Form
PropNounType	NPType
IVerbType	VPType = D → Form
TVerbType	NPType → VPType
DTVerbType	NPType → NPType → VPType

Meanings and Grammar

- Associate meanings w/production rules:

S → NP VP	{NP.sem(VP.sem)}
NP → Det Nom	{Det.sem(Nom.sem)}
NP → PropNoun	{PropNoun.sem}
Nom → Noun	{Noun.sem}
VP → IVerb	{IVerb.sem}
VP → TVerb NP	{TVerb.sem(NP.sem)}

Lexical Semantics

Det → every	{λP:NounType.λQ:VPType. ∀x.(P(x) ⇒ Q(x))}
Det → a	{λP: NounType.λQ:VPType. ∃x.(P(x) ∧ Q(x))}
Noun → chicken	{λx:D. *chicken*(x)}
PropNoun → Jane	{λP:VPType. P(*jane*)}
Verb → walked	{λx:D. *walked*(x)}
Verb → ate	{λo: NPType. λs:D. o(λy: D. *ate*(s,y))}

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

[[Every student read a book]]
= [[Every student]] ([[read a book]])

[[Every student]] = λQ.∀x(*student*(x)⇒Q(x))

[[read a book]] = λs:D. ∃y(*book*(y) ∧ *read*(s,y))

[[Every student]] ([[read a book]])
= ∀x(*student*(x)⇒(λs:D. ∃y(*book*(y) ∧ *read*(s,y)))(x))
= ∀x(*student*(x)⇒∃y(*book*(y) ∧ *read*(x,y)))

What about other meaning?

- Montague [1973]: Rewrite sentence:
  - A book, every student read it.
- "It" creates a hole to be filled:
  - [[every student read it]] = λz:D.∀x.(student(x) ⇒ read(x,z))
  - [[a book]] = λP.∃y.(book(y) ∧ P(y)) with type VPType → Form.

Putting it Together

- [[A book, every student read it]]
  $= (\lambda P.\exists y.(\text{book}(y) \wedge P(y)))$
  $(\lambda z{:}D.\forall x.(\text{student}(x) \Rightarrow \text{read}(x,z)))$
  $= \exists y.(\text{book}(y) \wedge (\lambda z{:}D.\forall x.(\text{student}(x) \Rightarrow \text{read}(x,z)))(y)))$
  $= \exists y.(\text{book}(y) \wedge \forall x.(\text{student}(x) \Rightarrow \text{read}(x,y)))$
- Seems like a trick!

Any Questions?