CS 181: Natural Language Processing

Lecture 17: Computational Semantics

Kim Bruce
Pomona College
Spring 2008

Disclaimer: Slide contents borrowed from many sources on web!

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]] = $\lambda Q.\forall x(\mathit{student}(x) \Rightarrow Q(x))$

[[read a book]] = $\lambda s{:}D.\ \exists y(\mathit{book}(y) \wedge \mathit{read}(s,y))$

[[Every student]] ([[read a book]])
= $\forall x(\mathit{student}(x) \Rightarrow (\lambda s{:}D.\ \exists y(\mathit{book}(y) \wedge \mathit{read}(s,y)))(x))$
= $\forall x(\mathit{student}(x) \Rightarrow \exists y(\mathit{book}(y) \wedge \mathit{read}(x,y)))$

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]] = $\lambda z{:}D.\forall x.(\text{student}(x) \Rightarrow \text{read}(x,z))$
  - [[a book]] = $\lambda P.\exists y.(\text{book}(y) \wedge P(y))$ with type VPType → Form.

Putting it Together

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

Other Solutions

- Cooper Storage:
  - "Freeze" meanings for quantifiers, pull out when needed. (*See book for similar idea*)
  - Results in saving multiple meanings.
- Doesn't work with nested noun phrases
  - "Jane read every book of a teacher."
  - Keller suggested an improvement
- Hole semantics incorporates constraints
  - Graphical representation representing constraints.

Semantic Ambiguity

- More attempted solutions:
  - Quasi-Logical Form, Underspecified Logical Form, Underspecified Discourse Representation Theory, Minimal Recursion Semantics, Ontological Promiscuity, Hole Semantics, the Constraint Language for Lambda Structures, and Normal Dominance Constraints
- Sentences w/N quantifiers have up to N! meanings.
- Desirable to return probability weightings

Logic in NLTK

Formulas

- >>> lp = nltk.sem.LogicParser()
- >>> lp.parse(r'(walk x)')
  - ApplicationExpression('walk', 'x')
- >>> lp.parse(r'\x.(walk x)')
  - LambdaExpression('x', '(walk x)')

Building formulas

- Examples:
  - >>> lp.parse('(and p q)')
    - ApplicationExpression('(and p)', 'q')
- *Allows infix*
  - >>> lp.parse('(p and q)')
    - ApplicationExpression('(and p)', 'q')
  - >>> e = lp.parse('(p and (not q))')
  - >>> e
    - ApplicationExpression('(and p)', '(not q)')

Formatting

- Examples:
  - >>> print e
    - (and p (not q))
  - >>> print e.infixify()
    - (p and (not q))

Using Lambda Calculus

- Examples
  - >>> e = lp.parse(r'(\x.((walk x) and (talk x)) john)')
  - >>> e
    - ApplicationExpression('\x.(and (walk x) (talk x))', 'john')
  - >>> e.simplify()
    - ApplicationExpression('(and (walk john))', '(talk john)')

Embedding FOL

- Examples:
  - >>> lp = nltk.sem.LogicParser(constants=['dog', 'walk', 'see'])
  - >>> lp.parse(r'dog')
    - ConstantExpression('dog')
  - >>> lp.parse('x')
    - IndVariableExpression('x')

Characteristic Functions

- Build characteristic functions from dicts:
  - >>> called = nltk.sem.CharFun({'bob': nltk.sem.CharFun({'mary':True}),
  - ... 'jane': nltk.sem.CharFun({'mary':True})})
  - >>> called['bob']
    - {'mary': True}
  - >>> called['bob']['mary']
    - True
  - >>> called['bob']['jane']
    - Traceback (most recent call last):
    - File "<stdin>", line 1, in <module>
    - KeyError: 'jane'

Building Characteristic Functions

- Build from relations:
  - >>> cd = set([('bob','mary'),('jane','sally')])
  - >>> cf = nltk.sem.CharFun()
  - >>> cf.read(cd)
  - >>> cf
    - {'sally': {'jane': True}, 'mary': {'bob': True}}

Building a Model

- Valuations interpret non-logical symbols:
  - >>> val = nltk.sem.Valuation({'JJ':'jane','Tiger':'bob','Mare':'mary', ..., 'boy':{'bob':True}, 'called':called})
  - >>> val['JJ']
    - 'jane'
  - >>> val['called']
    - {'jane': {'mary': True}, 'bob': {'mary': True}}

Building a Model

- Assignments:
  - >>> g = nltk.sem.Assignment(['jane','bob','mary','sally'], {'x':'bob','y':'mary'})

domain of model ↓

  - >>> g
    - {'y': 'mary', 'x': 'bob'}
  - >>> print g
    - g[mary/y][bob/x]
  - >>> g.add('sally','z')
    - {'y': 'mary', 'x': 'bob', 'z': 'sally'}

Building a Model

- Model:
  - m = nltk.sem.Model(set(['jane','bob','mary','sally']),val)
  - m.evaluate('some x.((boy x) and (called x Mare))',g)
    - True
    - *Searches over domain of model for satisfying assignment -- slow, as keeps going even after it finds it!*
    - *Warning: Tracing evaluate w/existentials raises exception at runtime!*

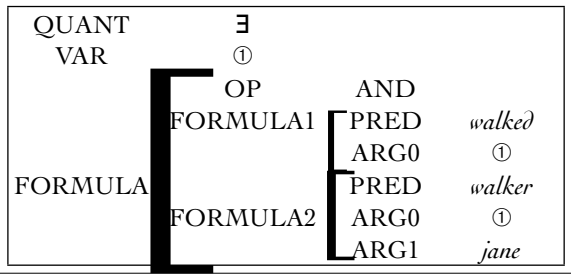
Unification Based Approaches

Adding Semantic Features

- Before associated non-CFG rules to productions:
  - NP → Det Nominal
    - <Det AGREEMENT> = < Nominal AGREEMENT>
    - <NP AGREEMENT> = <Nominal AGREEMENT>
- Want to associate semantics in same way.
- Give up FOL for now to explore use of feature constraints

Two Representations

- [[Jane walked]]
  = ∃e.*Walked(e)* ∧ *Walker(e,jane)*
- *or*



Example

- IVerb → walked
  - <IVerb SEM QUANT> = ∃
  - <IVerb SEM FORMULA OP> = ∧
  - <IVerb SEM FORMULA FORMULA1 PRED> = *walked*
  - <IVerb SEM FORMULA FORMULA1 ARG0> = <IVerb SEM VAR>
  - <IVerb SEM FORMULA FORMULA2 PRED> = *walker*
  - <IVerb SEM FORMULA FORMULA2 ARG0> = <IVerb SEM VAR>
  - <IVerb SEM FORMULA FORMULA2 ARG1> = <IVerb ARG0>

Draw picture!

Example

- VP → IVerb
  - <VP SEM> = <IVerb SEM>
  - <VP ARG0> = <IVerb ARG0>
- PropNoun → Jane
  - <PropNoun SEM PRED> = *Jane*
  - <PropNoun VAR> = <PropNoun SEM PRED>
- NP → PropNoun
  - <NP SEM> = <PropNoun SEM>
  - <NP SCOPE> = <PropNoun SCOPE>
  - <NP VAR> = <PropNoun VAR>

Put it all together …

- S → NP VP
  - <S Sem> = <NP Sem>
  - <VP ARG0> = <NP VAR>
  - <NP SCOPE> = <VP SEM>
- *Allows one to construct diagrams using unification as before.*
- *Can unify as parse as w/other features*

Semantics for Fragment of English

Back to Logic

- Handle declaratives, interrogatives, and imperatives.
- S → NP VP {DCL(NP.Sem(VP.sem))}
- S → VP {IMP(VP.sem(DummyYou))}
- S → Aux NP VP {YNQ(NP.Sem(VP.sem))}
- DCL, IMP, YNQ are operators resulting in actions.

Operators

- DCL
  - add to knowledge base
- YNQ
  - determine if prop can be inferred from knowledge base - meaning is correct answer
- IMP
  - speech act - discussed later on dialog

Wh-Questions

- Who ate the candy?
- S → WhWord VP {WHQ(WhWord.sem.var,VP.sem)}
  - λz:NP. z(VP.Sem) is function taking a NP.
  - Meaning is set of f in NP making fcn true.
- What did John eat?
- S → WhWord Aux NP VP
  - {WHQ(λo:NP(NP.sem((λx:NP: VP.sem(x))(o))} *or* {WHQ(o,NP.sem,VP.sem(x))}

Noun Phrases

- Already seen NPType = VPType → Form
- Nominal → Noun Nominal
  - {λx. Nominal.sem(x) ∧ NN(Noun.sem, x)}
  - summer vacation, head start, elephant gun, ...
  - Note that an elephant gun is not an elephant!

Adjective Phrases

- Try:
  - Nominal → Adj Nominal
    - {λx. Nominal.sem(x) ∧ IsA(x,Adj.sem)}
  - Adj → red {λP:NounType.λx.P(x) ∧ IsA(x,*red*)} ✓
  - Adj → small {λP:NounType.λx.P(x) ∧ IsA(x,*small*)} ✗
  - [[small mouse]] = λx. mouse(x) ∧ IsA(x,small)
  - [[small elephant]] = λx. elephant(x) ∧ IsA(x,small)
  - Others: former friend, fake gun, ...

Genitive Noun Phrases

- Mary's bike, Mary's friend, Ontario's airport.
  - Meaning determined more by noun, not possessive!
- NP → ComplexDet Nominal
- ComplexDet → NP's {NP.sem}
  - {λQ:VPType. ∃x. Nominal.sem(x) ∧ GN(x, ComplexDet.sem) ∧ Q(x)}
  - [[Mary's bike]] = λQ:VPType. (∃x. *bike(x)* ∧ *belongsTo(bike,Mary)* ∧ Q(x))

Prepositional Phrases

- Generalize from possessives:
  - Mary's bike ≡ the bike of Mary
  - [[house on a hill]]: NounType = D → Form
  - P → on {λP:NPType.λQ:NounType. λx:D. P(λy. Q(x) ∧ *on*(x,y))}
  - PP → P NP {P.sem(NP.sem)}
  - Nom → Nom PP {PP.sem(Nom.sem)}
- *Different from text!*

Semantics & Earley Parser

- Just like other features
- No extra problems!

Idioms

- Hard - don't make literal sense.
- Tip of the iceberg, Achilles' heel, ..
- Generally not compositional.
- Must recognize and treat separately

Summary of Semantics

- Principle of Compositionality key
- Syntax-driven semantic analysis
- Use λ-expressions (and other cheats)
- Quantifiers require lambda lifting.

Any Questions?