HW #6

Goals

- Semantics
  - Gain better understanding of semantic representations
  - Develop experience with lambda calculus and FOL
  - Create semantic attachments
  - Understand semantic composition

Compositional Semantics

- **Part 1**:
  - ***Manually*** create target semantic representations
  - Use Neo-Davidsonian event representation
    - e.g. verb representation with event variable, argument conjuncts
  - Can use as test cases for part 2

Compositional Semantics

- **Part 1**:
  - ***Manually*** create target semantic representations
  - Use Neo-Davidsonian event representation
    - e.g. verb representation with event variable, argument conjuncts
  - Can use as test cases for part 2
- **Part 2**:
  - Create semantic attachments to reproduce (NLTK)
  - Add to grammatical rules to derive sentence representations

Compositional Semantics

- **Part 1**:
  - ***Manually*** create target semantic representations
  - Use Neo-Davidsonian event representation
    - e.g. verb representation with event variable, argument conjuncts
  - Can use as test cases for part 2
- **Part 2**:
  - Create semantic attachments to reproduce (NLTK)
  - Add to grammatical rules to derive sentence representations
- Note: Lots of ambiguities (scope, etc)
  - Only need to produce one

Semantics in NLTK

- Grammar files:
  - .fcfg extension
  - Example format in NLTK Book Chapter 10
  - `/corpora/nltk/nltk-data/grammars/book_grammars/simple-sem.fcfg`
  - Note: Not “event-style”
- Parsing:
  - Use `nltk.parse.FeatureChartParser` (or similar)

Semantics in NLTK

- Printing semantic representations:

```
item.label()['SEM'].simplify()
    all x.(dog(x) -> exists e.(barking(e) & barker(e,x)))
```

- Also `nltk.sem.util.root_semrep(item)`

Semantic attachments in NLTK: Syntax

(The programming kind)

- `a,b,e,x`
  - lowercase variables can be arguments:
  - `\x.dog(x)`
- `P,Q,X`
  - uppercase lambda variables are functors
  - `\P.P(john)`

λ	=	`\`
∃	=	`exists`
∀	=	`all`
∧	=	`&`
∨	=	`|`
⇒	=	`->`
¬	=	`-`

More NLTK Logic Format

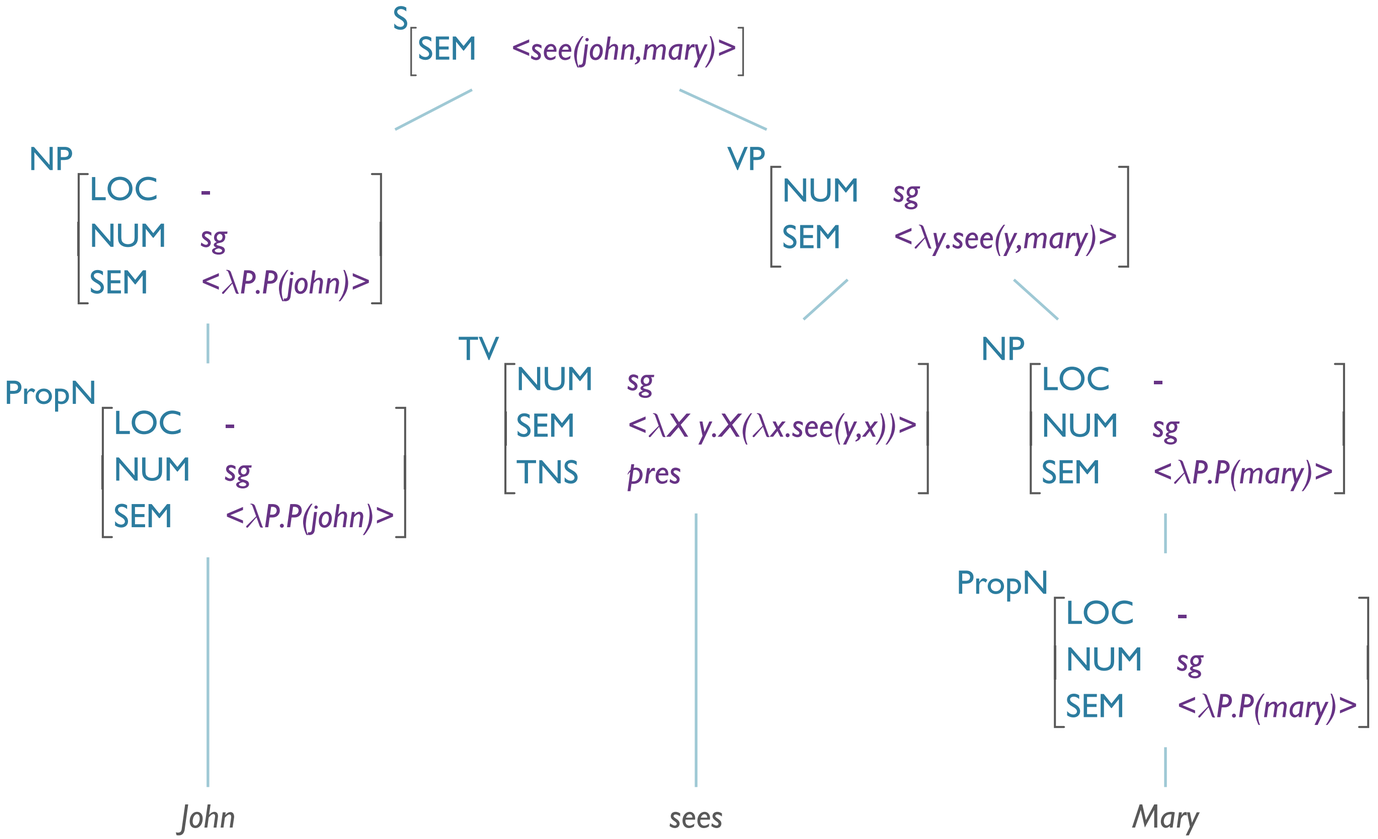
- Added to typical CFG rules
  - Basic approach similar to HW #5
  - Composing semantics:
    - `S[SEM=<?np(?vp)>] -> NP[SEM=?np] VP[SEM=?vp]`

More NLTK Logic Format

- Added to typical CFG rules
  - Basic approach similar to HW #5
  - Composing semantics:
    - `S[SEM=<?np(?vp)>] -> NP[SEM=?np] VP[SEM=?vp]`
- Creating lambdas:
  - `IV[SEM=<\x.exists e.(barking(e) & barker(e,x))>] -> 'barks'`

More NLTK Logic Format

- Added to typical CFG rules
  - Basic approach similar to HW #5
  - Composing semantics:
    - `S[SEM=<?np(?vp)>] -> NP[SEM=?np] VP[SEM=?vp]`
- Creating lambdas:
  - `IV[SEM=<\x.exists e.(barking(e) & barker(e,x))>] -> 'barks'`
- Nested lambdas:
  - `\x.\y. Etc → \x y.`
    Can remove '.' between sequences of lambda elements
    Keep '.' between sections: lambdas, quantifiers, body


S
SEM <see(john,mary)>
NP
LOC -
NUM sg
SEM <λP.P(john)>
VP
NUM sg
SEM <λy.see(y,mary)>
PropN
LOC -
NUM sg
SEM <λP.P(john)>
TV
NUM sg
SEM <λX y.X(λx.see(y,x))>
TNS pres
NP
LOC -
NUM sg
SEM <λP.P(mary)>
PropN
LOC -
NUM sg
SEM <λP.P(mary)>
John
sees
Mary