1 SPNLP 2008: From Syntax to Model Checking

Contents

2 Review

Logical Syntax and Semantics

- A logical language based on:
  1. function-argument structures: $(M\ N)$
  2. lambda abstraction: $\lambda x.(\alpha\, x)$
  3. beta-reduction: $(\lambda x.(M\ x)\ N) \equiv (M\ N)$
  4. Boolean combinations: $(\phi \wedge \psi), \ldots$
  5. Quantified formulas: $\forall x.\phi$, $\exists x.\phi$

- Models for the language:
  1. $M = \langle D, V \rangle$
  2. variable assignment $g : \mathit{Var} \mapsto D$
  3. recursive definition of $[\![\alpha]\!]^{M,g}$ for expressions α.
  4. $M, g \models \phi$ iff $[\![\phi]\!]^{M,g'} = 1$.

Compositional Semantics

Compositionality The meaning of a complex expression is a function of the meaning of its parts.

How do we know what the parts are?

- Feature-based context-free grammar formalism.
- Every category has a sem feature whose value is the semantics of expressions of that category:
  - lexical categories: fully-instantiated LF.

– phrasal categories: build an LF by function application over the LFs of the daughters.

Example PS Rule

```
S[sem = <app(?subj,?vp)>] -> NP[sem=?subj] VP[sem=?vp]
```

3 Computational Framework

Computational Recap

- Logical expressions are parsed into subclasses of `Expression` by `nltk.sem.logic`.
- Expressions can be evaluated in a model by `nltk.sem.evaluate`.
- English sentences can be parsed into LFs by `nltk.parse.featurechart` (*via* the `nltk.parse.load_earley(` function.)

Sample Interpretation

$$\begin{array}{ll} \textit{A dog barks} & \longrightarrow \\ \exists x.((\textit{dog}\, x) \wedge (\textit{bark}\, x)) & \longrightarrow \\ [\![\exists x.((\textit{dog}\, x) \wedge (\textit{bark}\, x)]\!]^{M,g} & = 1 \textit{iff} \ldots \end{array}$$

Parsing

```
import nltk
tokens = 'a dog barks'.split()
from nltk.parse import load_earley
cp = load_earley('grammars/sem1.fcfg', trace=0)
trees = cp.nbest_parse(tokens)
for t in trees:
    print t
```

Parsing Output

Parse for *A dog barks*

```
(S[sem=<some x.(and (dog x) (bark x))>]
  (NP[sem=<\P.some x.(and (dog x) (P x))>]
    (Det[sem=<\Q P.some x.(and (Q x) (P x))>] a)
    (N[sem=<dog>] dog))
  (VP[sem=<\x.(bark x)>]
    (IV[sem=<\x.(bark x)>] barks)))
```

Declaring a Model

Model for *A dog barks*

```
from nltk.sem import *
val = Valuation({
'fido': 'f',
'dog': {'f': True, 'd': True},
'bark': {'d': True},
})
dom = val.domain
m = Model(dom, val)
g = Assignment(dom)
```

Model Checking

Truth in model $\mathfrak{m}$

```
>>> print m
Domain = set(['d', 'f']),
Valuation =
{'bark': {'d': True},
'dog': {'d': True, 'f': True},
'fido': 'f'}
>>> g
{}
>>> m.evaluate('some x. ((dog x) and (bark x))', g)
True
```

Tracing

Truth in model $\mathfrak{m}$

```
>>> m.evaluate('some x.((dog x) and (bark x))',g,trace=1)

Open formula is '(and (dog x) (bark x))' with assignment g
  (trying assignment g[d/x])
  value of '(and (dog x) (bark x))' under g[d/x] is True
  (trying assignment g[f/x])
  value of '(and (dog x) (bark x))' under g[f/x] is False
 '(and (dog x) (bark x))' evaluates to True under M, g
'some x. ((dog x) and (bark x))' evaluates to True under M, g
```

4 Alternative Input Formats for Valuations

Inputting Valuations: Vanilla Method

```
from nltk.sem import *
val = Valuation({
'fido': 'f',
'kim': 'k'
'chase': {'f': {'k': True},
          'k': {'f': True}}
})
dom = val.domain
m = Model(dom, val)
g = Assignment(dom)
```

Inputting Valuations: Read in tuples

```
from nltk.sem import *
val = Valuation()
v = [('fido', 'f'),
     ('kim', 'k'),
     ('chase', set([('f', 'k'), ('k', 'f')]))
    ]
val.read(v)
dom = val.domain
m = Model(dom, val)
g = Assignment(dom)
```

Inputting Valuations: Read from string (or file)

```
from nltk.sem import *
v = """
   fido => f
   kim => k
   chase => {(f, k), (k, f)}
"""
val = parse_valuation(v)
dom = val.domain
m = Model(dom, val)
g = Assignment(dom)
```

5 Getting the Output

Examining Valuations

Outputting tuples

```
>>> val
{'f': 'f', 'kim': 'k',
 'chase': {'k': {'f': True}, 'f': {'k': True}}}
>>> relation = val['chase']
```

```
>>> relation
{'k': {'f': True}, 'f': {'k': True}}
>>> relation.tuples()
set([('k', 'f'), ('f', 'k')])
>>> val['run']
Traceback (most recent call last):
...
nltk.sem.evaluate.Undefined: Unknown expression: 'run'
>>> m.evaluate('\\x. (chase x kim)', g)
{'f': True}
>>> m.evaluate('\\x. some y. (chase x y)', g).tuples()
set(['k', 'f'])
```

Mapping from Syntax to Semantics, 1

Parse sentence & load valuation

```
from nltk.parse import FeatureEarleyChartParser
import nltk.data
grammar = nltk.data.load('grammars/sem2.fcfg')
val = nltk.data.load('grammars/valuation1.val')
dom = val.domain
m = Model(dom, val)
g = Assignment(dom)
sent = 'some girl chases a dog'
result = nltk.sem.text_evaluate([sent], grammar, m, g)
for (syntree, semrep, value) in result[sent]:
   print "'%s' is %s in Model m\n" % (semrep.infixify(), value)
```

Mapping from Syntax to Semantics, 2

Result

```
'some x.((girl x) and
         some z559.((dog z559) and
                    (chase z559 x)))'
is True in Model m
```

6 Summary

Summary

- The NLTK implementation yields an end-to-end mapping:
  - Compute all parses of a sentence S relative to a feature-based CFG;
  - provide a logical form for each constituent of S;
  - parse the logical form LF for each reading of S;
  - build a representation of a first order model M;
  - recursively evaluate LF in M.

  - If LF contains free variables, then value also depends on g.
- Major shortcoming so far: no treatment of *semantic* ambiguity, e.g., quantifier scope ambiguity.
- Two approaches in `nltk.contrib`: `hole.py` and `gluesemantics` package.