Lab 3: Semantics

TDT4275: Natural Language Interfaces

Jonas Myrlund

April 9, 2013

1 Written assignments

1.1 Feature-based grammars

Documentation exists alongside code. Please see `feat1.fcfg` for details.

1.1.1 Example runs

The packaged tool is simple to use, and to try out new sentences with. From the root folder, run `python run.py --help` for an explanation.

The default output of the command outputs the following:

```
I want to spend lots of money            OK
me want to spend lots of money           FAIL

tell me about Chez Parnisse              OK
tell I about Chez Parnisse               FAIL

I would like to take her out to dinner   OK
I would like to take she out to dinner   FAIL

she does not like Italian                OK
her does not like Italian                FAIL

this dog runs                            OK
I runs                                   FAIL
these dogs runs                          FAIL
```

To run a specific sentence through the parser, the command line flag `-s/--sentence` is utilized. To display detailed trace information and generated parse trees, the `--debug` flag can be used.

The command `python run.py --sentence ‘‘I want to spend lots of money’’ --debug` thus yields:

```
|.I.w.t.s.l.o.m.|
Leaf Init Rule:
|[-] . . . . . .| [0:1] 'I'
|. [-] . . . . .| [1:2] 'want'
|. . [-] . . . .| [2:3] 'to'
|. . . [-] . . .| [3:4] 'spend'
|. . . . [-] . .| [4:5] 'lots'
|. . . . . [-] .| [5:6] 'of'
|. . . . . . [-]| [6:7] 'money'
Feature Bottom Up Predict Combine Rule:
|[-] . . . . . .| [0:1] Pro[Form='sub', Num='sg', Per=1] -> 'I' *
Feature Bottom Up Predict Combine Rule:
|[-> . . . . . .| [0:1] S[] -> Pro[Form='sub', Num=?n] * VP[Num=?n,
Feature Bottom Up Predict Combine Rule:
|. [-] . . . . .| [1:2] V[Tense='inf', Type='trans'] -> 'want' *
Feature Bottom Up Predict Combine Rule:
|. [-> . . . . .| [1:2] S[] -> V[Tense='inf'] * Pro[Form='obj'] PP[]
|. [-> . . . . .| [1:2] VP[Num=?n, Tense=?t] -> V[Num=?n, Tense=?t,
|. [-> . . . . .| [1:2] VP[Num=?n, Tense=?t] -> V[Num=?n, Tense=?t,
|. [-] . . . . .| [1:2] V[Num='pl', Tense='pres'] -> V[Tense='inf']
Feature Bottom Up Predict Combine Rule:
|. [-] . . . . .| [1:2] VP[Num='pl', Per=?p, Tense='pres'] -> V[Num=
|. [-> . . . . .| [1:2] VP[Num=?n, Tense=?t] -> V[Num=?n, Tense=?t,
|. [-> . . . . .| [1:2] VP[Num=?n, Tense=?t] -> V[Num=?n, Tense=?t,
|. [-> . . . . .| [1:2] VP[Num=?n, Tense=?t] -> V[Num=?n, Tense=?t,
Feature Bottom Up Predict Combine Rule:
|. . [-] . . . .| [2:3] Inf[] -> 'to' *
Feature Bottom Up Predict Combine Rule:
|. . [-> . . . .| [2:3] VP[Tense='inf', +inf] -> Inf[] * VP[Tense='i
|. . [-] . . . .| [2:3] Aux[] -> Inf[] *
Feature Bottom Up Predict Combine Rule:
|. . [-> . . . .| [2:3] VP[+aux] -> Aux[] * VP[Tense='inf'] {}
|. . [-> . . . .| [2:3] Aux[] -> Aux[] * 'not' {}
Feature Bottom Up Predict Combine Rule:
|. . . [-] . . .| [3:4] V[Tense='inf', Type='trans'] -> 'spend' *
Feature Bottom Up Predict Combine Rule:
|. . . [-> . . .| [3:4] S[] -> V[Tense='inf'] * Pro[Form='obj'] PP[]
|. . . [-> . . .| [3:4] VP[Num=?n, Tense=?t] -> V[Num=?n, Tense=?t,
|. . . [-> . . .| [3:4] VP[Num=?n, Tense=?t] -> V[Num=?n, Tense=?t,
|. . . [-] . . .| [3:4] V[Num='pl', Tense='pres'] -> V[Tense='inf']
Feature Bottom Up Predict Combine Rule:
|. . . [-] . . .| [3:4] VP[Num='pl', Per=?p, Tense='pres'] -> V[Num=
|. . . [-> . . .| [3:4] VP[Num=?n, Tense=?t] -> V[Num=?n, Tense=?t,
|. . . [-> . . .| [3:4] VP[Num=?n, Tense=?t] -> V[Num=?n, Tense=?t,
|. . . [-> . . .| [3:4] VP[Num=?n, Tense=?t] -> V[Num=?n, Tense=?t,
Feature Bottom Up Predict Combine Rule:
|. . . . [-> . .| [4:5] N[Num='mass'] -> 'lots' * 'of' 'money' {}
Feature Single Edge Fundamental Rule:
|. . . . [---> .| [4:6] N[Num='mass'] -> 'lots' 'of' * 'money' {}
Feature Single Edge Fundamental Rule:
```

```
|. . . . [-----]| [4:7] N[Num='mass'] -> 'lots' 'of' 'money' *
Feature Bottom Up Predict Combine Rule:
|. . . . [-----]| [4:7] NP[Num='mass'] -> N[Num='mass'] *
Feature Bottom Up Predict Combine Rule:
|. . . . [----->| [4:7] S[] -> NP[Num=?n] * VP[Num=?n, -inf] {?n: 'm
Feature Single Edge Fundamental Rule:
|. . . [-------]| [3:7] VP[Num=?n, Tense='inf'] -> V[Num=?n, Tense='
|. . . [-------]| [3:7] VP[Num='pl', Tense='pres'] -> V[Num='pl', Te
Feature Single Edge Fundamental Rule:
|. . [---------]| [2:7] VP[Tense='inf', +inf] -> Inf[] VP[Tense='inf
|. . [---------]| [2:7] VP[+aux] -> Aux[] VP[Tense='inf'] *
Feature Single Edge Fundamental Rule:
|. [-----------]| [1:7] VP[Num=?n, Tense='inf'] -> V[Num=?n, Tense='
|. [-----------]| [1:7] VP[Num='pl', Tense='pres'] -> V[Num='pl', Te
Feature Single Edge Fundamental Rule:
|[=============]| [0:7] S[] -> Pro[Form='sub', Num='sg'] VP[Num='sg'
Feature Single Edge Fundamental Rule:
|. [-----------]| [1:7] VP[Num=?n, Tense='inf'] -> V[Num=?n, Tense='
|. [-----------]| [1:7] VP[Num='pl', Tense='pres'] -> V[Num='pl', Te
I want to spend lots of money            OK

(S[]
  (Pro[Form='sub', Num='sg', Per=1] I)
  (VP[Num=?n, Tense='inf']
    (V[Tense='inf', Type='trans'] want)
    (VP[Tense='inf', +inf]
      (Inf[] to)
      (VP[Num=?n, Tense='inf']
        (V[Tense='inf', Type='trans'] spend)
        (NP[Num='mass'] (N[Num='mass'] lots of money))))))

(S[]
  (Pro[Form='sub', Num='sg', Per=1] I)
  (VP[Num=?n, Tense='inf']
    (V[Tense='inf', Type='trans'] want)
    (VP[+aux]
      (Aux[] (Inf[] to))
      (VP[Num=?n, Tense='inf']
        (V[Tense='inf', Type='trans'] spend)
        (NP[Num='mass'] (N[Num='mass'] lots of money))))))
```

1.2 First Order Logic

FOL-expressions for sentences:

Sharks do not eat birds	$\forall x, y\,(Shark(x) \wedge Bird(y) \wedge \neg Eats(x, y))$
Not all birds lay eggs	$\neg(\forall x\,(Bird(x) \wedge LaysEggs(x))$

1.2.1 NLTK-format

Sharks do not eat birds	`all x y.(Shark(x) & Bird(y) & -Eats(x, y))`
Not all birds lay eggs	`-(all x.(Bird(x) & LaysEggs(x)))`

1.2.2 Running the expressions

The expressions run nicely through the NLTK Logic Parser. Calling `free()` on the resulting objects yields empty sets, as expected.

Free variables, contrary to *bound variables*, means that the variable is not associated with a quantifier, such as $\forall$ or $\exists$. In the case of the above expressions, all cases of variables are both bound to $\forall$-quantifiers.

1.2.3 World models

The following code builds a simple set of logical expressions:

```
lp = nltk.LogicParser()

a = lp.parse('exists x.(samfundet(x) and school(x))')
b = lp.parse('smart(jonas)')
c = lp.parse('-smart(jonas)')
```

To execute them together, we can run them in the following manner:

```
mace = nltk.Mace()

mace.build_model(None, [a, b]) # => True
mace.build_model(None, [a, c]) # => True
mace.build_model(None, [b, c]) # => False
```

2 Lambda-based semantics

Exercise A

...

Exercise B

I was unable to get both distinct parses of the "every" determiner working. I simply added the following pair of rules to get a primitive version:

```
Det[NUM=sg,SEM=<\P Q.all x.(P(x) & Q(x))>] -> 'every'
Det[NUM=pl,SEM=<\P Q.exists x.(P(x) & Q(x))>] -> 'some'
```