

A Brief Incomplete Introduction to NLTK

This introduction ignores and simplifies many aspects of the Natural Language Toolkit, focusing on implementing and using simple context-free grammars and lexicons. Much more information can be found in the NLTK documentation, which is at <http://nltk.sourceforge.net/index.php/Documentation>, and the official NLTK web site <http://nltk.sourceforge.net/>, where you can download the code.

As stated on the web site:

NLTK—the Natural Language Toolkit—is a suite of open source Python modules, data and documentation for research and development in natural language processing. NLTK contains code supporting dozens of NLP tasks, along with 30 popular corpora and extensive documentation including a 360-page online book. Distributions for Windows, Mac OSX and Linux are available.

The NLTK book contains everything you want to know in a manner that is difficult to comprehend and navigate. This introduction provides a few examples that will hopefully be enough for you to implement and test simple grammars.

Setup

NLTK is implemented in Python, which is a scripting language comparable to Perl, though Python is more object-oriented and syntactically consistent than Perl. The most distinctive feature of Python is that, instead of using curly braces or some other begin-end tokens for block structure, it uses *indentation*. For example, all the statements in a for loop need to be indented the same number of spaces (inner loops or statements inside ifs are further indented, of course). The Python interpreter available on the Linux and Window machines is suitable for NLTK. The Sun machines appear to have an older version of Python which is not suitable.

On the Linux machines, you need to tell Python where the NLTK packages are:

```
setenv PYTHONPATH /home/bylander/nltk-0.8/build/lib
or
PYTHONPATH=/home/bylander/nltk-0.8/build/lib
```

I think the Windows machines have NLTK installed. Note that we are using version 0.8. Version 0.9 is now available, but I have not tested it.

NLTK Lexicons and Grammars

Here is the wumpus lexicon from the book (Figure 22.3) implemented for NLTK:

```
# #####
# Lexical Rules
# #####

N -> 'stench' | 'breeze' | 'glitter' | 'nothing' | 'agent'
```

```

N -> 'wumpus' | 'pit' | 'pits' | 'gold' | 'east'

V -> 'is' | 'see' | 'smell' | 'shoot' | 'feel' | 'stinks' | 'go'
V -> 'grab' | 'carry' | 'kill' | 'turn'

Adj -> 'right' | 'left' | 'east' | 'dead' | 'back' | 'smelly'

Adv -> 'here' | 'there' | 'nearby' | 'ahead' | 'right' | 'left'
Adv -> 'east' | 'south' | 'back'

PN -> 'me' | 'you' | 'I' | 'it'

PropN -> 'John' | 'Mary' | 'Boston' | 'UCB' | 'Aristotle'

Det -> 'the' | 'a' | 'an'

Prep -> 'to' | 'in' | 'on' | 'near'

Conj -> 'and' | 'or' | 'but'

Digit -> '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

That -> 'that'

```

I have abbreviated/changed some of the category names, such as *Name* with PropN (proper name) and *Article* with Det (determiner). I have also added a category That for relative clauses.

Here is the wumpus grammar from the book (Figure 22.4) implemented for NLTK:

```

% start S
# #####
# Grammar Rules
# #####

# S expansion rules
S -> NP VP
S -> S Conj S

# NP expansion rules
NP -> PN
NP -> PropN
NP -> N
NP -> Det N
NP -> Digit Digit
NP -> NP PP
NP -> NP RelClause

```

```

# VP expansion rules
VP -> V
VP -> VP NP
VP -> VP Adj
VP -> VP PP
VP -> VP Adv

# misc expansion rules
PP -> Prep NP
RelClause -> That VP

```

S is declared to be the start symbol. The other rules are straightforward from the book. This grammar with the lexicon can be downloaded from my web site.

Other examples of grammars can be found in `/home/bylander/nltk-0.8/examples`. One of them will be described in more detail below.

Using a Grammar in NLTK

The Python interpreter can be used to load and run a grammar. You should have an `nltk` directory which contains the above `wumpus.cfg` context-free grammar. In the commands below, what I've typed is underlined. Make sure you are connected to your `nltk` directory first.

- Set the `PYTHONPATH` environmental variable:

```
main203{bylander}1: setenv PYTHONPATH /home/bylander/nltk-0.8/build/lib
```

- Run the Python interpreter:

```
main203{bylander}2: python
Python 2.4.3 (#2, Oct 6 2006, 07:52:30)
[GCC 4.0.3 (Ubuntu 4.0.3-1ubuntu5)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

- Import everything in NLTK.

```
>>> from nltk import *
```

- Read the context-free grammar.

```
>>> g = parse.GrammarFile('wumpus.cfg')
```

- Prepare a parser using the CFG. Use `trace=1` or `trace=2` if you want some output later to puzzle over.

```
>>> p = g.earley_parser(trace=0)
```

- Prepare a sentence for parsing.

```
>>> s = list(tokenize.whitespace('the wumpus is on 2 2'))
```

- Get the possible parse trees for the sentence.

```
>>> trees = p.get_parse_list(s)
```

- Print the parse trees. You will need to press enter twice. `margin=0` forces one tree node per line.

```
>>> for tree in trees: print(tree.pprint(margin=0))
```

You can make life a little simpler for yourself by using the `get_parser` and `do_parse` Python commands in `nltk_stuff.py` available from my web site. You need to put this file in your `nltk` directory. Then you can do the following commands in Python:

```
from nltk import *
from nltk_stuff import *
p = get_parser('wumpus.cfg', trace=1)
do_parse('the wumpus is on 2 2', p, margin=0)
```

Implementing Semantics in NLTK

To understand how a semantic interpretation can be obtained in NLTK, the example grammar `sem2.cfg` will be used. This example will also show how a grammar can be augmented with features so that rules for subject-verb agreement can be implemented.

Here is the lexicon from this example:

```
#####
# Lexical Rules
#####

PropN[-loc,num=sg,sem=<\P.(P john)>] -> 'John'
PropN[-loc,num=sg,sem=<\P.(P mary)>] -> 'Mary'
PropN[-loc,num=sg,sem=<\P.(P suzie)>] -> 'Suzie'
PropN[-loc,num=sg,sem=<\P.(P fido)>] -> 'Fido'
PropN[+loc, num=sg,sem=<\P.(P noosa)>] -> 'Noosa'

NP[-loc, num=sg, sem=<\P.\x.(P x)>] -> 'who'

Det[num=sg,sem=<\P Q. all x. ((P x) implies (Q x))>] -> 'every'
Det[num=pl,sem=<\P Q. all x. ((P x) implies (Q x))>] -> 'all'
Det[sem=<\P Q. some x. ((P x) and (Q x))>] -> 'some'
Det[num=sg,sem=<\P Q. some x. ((P x) and (Q x))>] -> 'a'

N[num=sg,sem=<boy>] -> 'boy'
N[num=pl,sem=<boy>] -> 'boys'
N[num=sg,sem=<girl>] -> 'girl'
```

$N[\text{num}=\text{pl}, \text{sem}=\langle \text{girl} \rangle] \rightarrow \text{'girls'}$
 $N[\text{num}=\text{sg}, \text{sem}=\langle \text{dog} \rangle] \rightarrow \text{'dog'}$
 $N[\text{num}=\text{pl}, \text{sem}=\langle \text{dog} \rangle] \rightarrow \text{'dogs'}$

$TV[\text{num}=\text{sg}, \text{sem}=\langle \lambda X y. (X \ \backslash x. (\text{chase } x \ y)) \rangle, \text{tns}=\text{pres}] \rightarrow \text{'chases'}$
 $TV[\text{num}=\text{pl}, \text{sem}=\langle \lambda X y. (X \ \backslash x. (\text{chase } x \ y)) \rangle, \text{tns}=\text{pres}] \rightarrow \text{'chase'}$
 $TV[\text{num}=\text{sg}, \text{sem}=\langle \lambda X y. (X \ \backslash x. (\text{see } x \ y)) \rangle, \text{tns}=\text{pres}] \rightarrow \text{'sees'}$
 $TV[\text{num}=\text{pl}, \text{sem}=\langle \lambda X y. (X \ \backslash x. (\text{see } x \ y)) \rangle, \text{tns}=\text{pres}] \rightarrow \text{'see'}$
 $TV[\text{num}=\text{sg}, \text{sem}=\langle \lambda X y. (X \ \backslash x. (\text{chase } x \ y)) \rangle, \text{tns}=\text{pres}] \rightarrow \text{'chases'}$
 $TV[\text{num}=\text{pl}, \text{sem}=\langle \lambda X y. (X \ \backslash x. (\text{chase } x \ y)) \rangle, \text{tns}=\text{pres}] \rightarrow \text{'chase'}$
 $IV[\text{num}=\text{sg}, \text{sem}=\langle \lambda x. (\text{bark } x) \rangle, \text{tns}=\text{pres}] \rightarrow \text{'barks'}$
 $IV[\text{num}=\text{pl}, \text{sem}=\langle \lambda x. (\text{bark } x) \rangle, \text{tns}=\text{pres}] \rightarrow \text{'bark'}$
 $IV[\text{num}=\text{sg}, \text{sem}=\langle \lambda x. (\text{walk } x) \rangle, \text{tns}=\text{pres}] \rightarrow \text{'walks'}$
 $IV[\text{num}=\text{pl}, \text{sem}=\langle \lambda x. (\text{walk } x) \rangle, \text{tns}=\text{pres}] \rightarrow \text{'walk'}$

$P[+\text{loc}, \text{sem}=\langle \lambda X P x. (X \ \backslash y. ((P \ x) \ \text{and} \ (\text{in } y \ x))) \rangle] \rightarrow \text{'in'}$
 $P[-\text{loc}, \text{sem}=\langle \lambda X P x. (X \ \backslash y. ((P \ x) \ \text{and} \ (\text{with } y \ x))) \rangle] \rightarrow \text{'with'}$

Here is an explanation of a few of these lexical rules.

- $N[\text{num}=\text{pl}, \text{sem}=\langle \text{boy} \rangle] \rightarrow \text{'boys'}$
 'boys' is a plural ($\text{num}=\text{pl}$) noun. Its semantics is boy ($\text{sem}=\langle \text{boy} \rangle$), which will be used as a predicate.
- $\text{PropN}[-\text{loc}, \text{num}=\text{sg}, \text{sem}=\langle \lambda P. (P \ \text{mary}) \rangle] \rightarrow \text{'Mary'}$
 'Mary' is a proper noun, is not a location ($-\text{loc}$), is singular ($\text{num}=\text{sg}$), and has semantics $\lambda P. (P \ \text{mary})$. The semantics represent the lambda expression $\lambda P \ P(\text{mary})$. P is a predicate to be filled in later.
- $IV[\text{num}=\text{sg}, \text{sem}=\langle \lambda x. (\text{walk } x) \rangle, \text{tns}=\text{pres}] \rightarrow \text{'walks'}$
 'walks' is an intransitive verb, is singular, is present tense, and has semantics $\lambda x. (\text{walk } x)$. The semantics represent the lambda expression $\lambda x \ \text{walk}(x)$. x is a term to be filled in later.

The semantics for 'Mary walks' can be obtained by two lambda applications.

Apply $\lambda P \ P(\text{mary})$ to $\lambda x \ \text{walk}(x)$. That is, the second expression will be used to fill in the P argument of the first expression.

The result is $(\lambda x \ \text{walk}(x)) (\text{mary})$. Here the lambda expression $\lambda x \ \text{walk}(x)$ is being applied to mary . That is, mary will be used to fill in the x argument of the lambda expression.

The final result is $\text{walk}(\text{mary})$.

The reason for the complication is so that more sophisticated semantic expressions can be constructed.

- `Det[sem=<\P Q. some x. ((P x) and (Q x))>] -> 'some'`

The semantics correspond to the lambda expression $\lambda P \lambda Q \exists x (P(x) \wedge Q(x))$. The idea is that one predicate comes from the noun (e.g., 'boys') and that the other predicate comes from the verb (e.g., 'walk'). We want to end up with something like $\exists x (\text{boy}(x) \wedge \text{walk}(x))$. For the word 'some', the existential quantifier is known and the lambda expression allows the inside to be filled in later.

As can be seen, the semantics of the lexicon set up lambda expressions so that a semantic interpretation is constructed when they are applied to each other. The grammatical rules (which also ensure feature agreement) specify the order of the lambda applications when there are two expressions to be combined.

```
% start S
#####
# Grammar Rules
#####

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

NP[num=?n,sem=<app(?det,?nom)> ] -> Det[num=?n,sem=?det] Nom[num=?n,sem=?nom]
NP[loc=?l,num=?n,sem=?np] -> PropN[loc=?l,num=?n,sem=?np]

Nom[num=?n,sem=?nom] -> N[num=?n,sem=?nom]
Nom[num=?n,sem=<app(?pp,?nom)>] -> N[num=?n,sem=?nom] PP[sem=?pp]

VP[num=?n,sem=<app(?v,?obj)>] -> TV[num=?n,sem=?v] NP[sem=?obj]
VP[num=?n,sem=?v] -> IV[num=?n,sem=?v]

VP[num=?n,sem=<app(?pp,?vp)>] -> VP[num=?n,sem=?vp] PP[sem=?pp]

PP[sem=<app(?p,?np)>] -> P[loc=?l,sem=?p] NP[loc=?l,sem=?np]
```

Using Semantics in NLTK

Constructing semantic interpretations in NLTK requires a different sequence of commands in Python.

- Import everything in NLTK.

```
>>> from nltk import *
```

- Read a file containing a context-free grammar with semantics

```
>>> g = parse.GrammarFile('sem2.cfg')
```

- Create a sentence to be interpreted.

```
>>> s = 'Suzie chases a dog'
```

- Produce interpretations. Use `syntrace = 1` to see some output to ponder.

```
>>> r = sem.text_interpret([s], g, syntrace=0)
```

- Print the trees and their interpretations.

```
for pair in r[s]:  
    (tree, interp) = pair  
    print tree  
    print interp
```

In the end, you should see the interpretation:

```
some x.(and (dog x) (chase x suzie))
```

If you installed NLTK on your own computer and you get something like `(?subj ?vp)`, then you need to replace `featstruct.py` in your NLTK installation with the hacked version on my web site.

Here is how the semantic information in the lexicon gets combined to obtain this result. As needed, parentheses are added to delimit expressions.

- Here are the meanings of the words:

```
'Suzie'   \P.(P suzie)  
'chases' \X y.(X \x.(chase x y)  
'a'      \P Q.some x.(and (P x) (Q x))  
'dog'    dog
```

- The meaning of 'a' is applied to the meaning of 'dog'.

```
'a dog'  (\P Q.some x.(and (P x) (Q x))) (dog)  
        \Q.some x.(and (dog x) (Q x))
```

- The meaning of 'chases' is applied to the meaning of 'a dog'. One of the `x` variables has been changed to `z` to reduce confusion.

```
'chases a dog' (\X y.(X \z.(chase z y)) (\Q.some x.(and (dog x) (Q x)))  
              (\y. ((\Q.some x.(and (dog x) (Q x))) (\z.(chase z y))))  
              \y. some x.(and (dog x) ((\z.(chase z y)) x))  
              \y. some x.(and (dog x) (chase x y))
```

- Finally, the meaning of 'Suzie' is applied to the meaning of 'chases a dog'.

```
'Suzie chases a dog' (\P.(P suzie)) (\y. some x.(and (dog x) (chase x y)))  
                        (\y. some x.(and (dog x) (chase x y))) (suzie)  
                        some x.(and (dog x) (chase x suzie))
```

Python commands in `nltk_stuff.py` make this a little bit simpler:

```
from nltk import *
from nltk_stuff import *
g = get_grammar('sem2.cfg')
do_semantics('Suzie chases a dog', g, trace=0)
```

With the augmentations to the grammar and the processing that goes with it, this style of parsing and semantics provides an initial approximation to parsing sentences and deriving their meanings. As you might guess, actual speech and text are not so grammatically and logically constrained as we might like them to be. Probabilistic approaches provide a way to be more flexible.