Lab for week 5: Recursive Descent Parser -- Answers

Adding productions to the grammar

• What production should be added to handle the sentence "He ate salad"?

Add the production \texttt{Pro -> 'He'}

• Is there a problem with either of the parse trees?

The parse tree for "he ate salad with a fork" is correct: the preposition phrase "with a fork" is correctly attached to the verb ('high' attachment). For the sentence "he ate salad with mushrooms", the preposition phrase "with mushrooms" is wrongly attached to the verb, whereas it should be attached to the noun phrase "salad" ('low' attachment).

• Change the order of the rules "NP -> N" and "NP -> NP PP"

This leads into a standard problem of left recursion as "NP -> NP PP" production is applied infinitely.

• How do you think this behaviour depends on the particular way this recursive descent parser chooses which rule to expand when there are multiple options?

Clearly it does: the infinite recursion depends on the left-recursive rule being chosen before other options (although note that in the case of some \textit{unparseable} strings the recursion would happen no matter what the order was.

Ungrammatical sentences

• Though the second sentence is ungrammatical, it parsed by our grammar. Modify the grammar to handle such cases

\texttt{Change VP -> V | V NP | V NP PP to VP -> Vi | Vt NP | Vp NP PP. This will exploit the subcategorization information in the alternative set of productions for the verbs.}

Number agreement (optional)

Change the grammar to handle number agreement and parse the following sentences:

\texttt{Use grammar2 in lab5-sol.py}

\footnote{http://creativecommons.org/licenses/by-nc/4.0/}
\footnote{lab5-sol.py}
Exploring a treebank grammar

- What is the type of the parsed sentence object (Hint: `type` command)?

  ```python
type(psents[0])
gives the object type which is `nltk.tree.Tree`
```

- Extract the list of words and the list of word,pos-tag tuples from `psents[0]` using some of the other available methods.

  ```python
  psents[0].leaves() and psents[0].pos()' will give, respectively, the list of words and word, pos-tag tuples
  ```

Distribution of Productions

- What are the 10 most frequent and least frequent lexical and grammatical productions?

  First download the answer code and look at the definition of `production_distribution`

  ```python
  Then do %run lab5-sol.py
  lex_prods, nonlex_prods = production_distribution(psents)
  ```

  For the 10 most frequent productions

  ```python
  sorted(lex_prods.items(), key=lambda x : x[1], reverse=True)[:10]
  sorted(nonlex_prods.items(), key=lambda x : x[1], reverse=True)[:10]
  ```

  10 least frequent productions

  ```python
  sorted(lex_prods.items(), key=lambda x : x[1])[:10]
  sorted(nonlex_prods.items(), key=lambda x : x[1])[:10]
  ```

Going further

1. What is the percentage of the 10 most frequent grammatical and lexical productions, with respect to the total number of productions?

   ```python
   100*sum(sorted(lex_prods.values(),reverse=True)[10:])//sum(lex_prods.values())
   100*sum(sorted(nonlex_prods.values(),reverse=True)[10:])//sum(nonlex_prods.values())
   ```

2. Run the parser on the sentence "John ate salad". Add required grammatical and lexical productions to handle this sentence.

   ```python
   We should add rules to handle proper nouns as below
   NP -> PropN ; PropN -> 'John'
   ```

3. Does the rule-ordering trick work if you add a rule of the form: `Det -> NP ‘s’`

   This leads into the left recursion problem again, so is only OK in last position, and again will still lead to infinite recursion with (some) ungrammatical input (try the `fork`).

4. The grammar which handles number agreement still parses ungrammatical sentences like "i sleeps". Adding more agreement rules is a messy process. What is the better way to deal with this problem?

   ```python
   We can generalise the rules using features.
   ```

5. After extracting the distributions from the treebank
lps = sorted(lex_prods.items(), key=lambda x: x[1], reverse=True)
plot_histogram(lps[:50])

nlps = sorted(nonlex_prods.items(), key=lambda x: x[1], reverse=True)
plot_histogram(nlps[:50])

Remember to expand the plot window so you can see what’s going on, and to close it when you want to get back to ipython.