Inf2A – Tutorial Exercise 7
SR PARSING, CHART PARSING AND PROBABILISTIC CFGs
Week 8 (9–13 November 2009)

1 Shift Reduce Parsing

In this section, we will look in more detail at the search strategy used in shift-reduce (SR) parsing. (Recall from Lecture 13 that all parsers have both a directionality and a search strategy. The directionality of SR parsing is simply bottom up.)

We will use the NLTK demo of shift-reduce parsing:

```python
>>> from nltk.app import srparser
>>> srparser()
```

Recall from class that in the demo, the parser fails to get any parses for the (grammatical, globally ambiguous) sentence

```
My dog saw a man in the park with a statue
```

(While we have already stepped through the whole demo in class, you might want to do so again – using the button marked “step” – to remind yourself of how it fails.)

What we’ll be looking at here is how the SR parser explores the search space of possible analyses and what this follows from. In doing so, you’ll be figuring out (without using back-tracking, which isn’t implemented in this version of SR parsing) how we can get at least one parse of this sentence.

1. First explore keeping the grammar as is, and changing the parsing strategy (which we can call eager reduction). Step through the demo until the parser builds (through reduction) an NP from and man. This is where eager reduction leads to failure. From here on, instead of stepping through the built-in strategy, choose at each point, whether to shift or reduce. Can you come up with a sequence of shifts and reductions that succeeds in getting a complete parse? Which one? Can you get them all? How would you characterise your strategy? Would you call it eager shifting (ie, shift whenever possible, and reduce only when you can’t shift)? conditional eager reduction (ie, always reduce under certain conditions, but not always)? Something else? Can you see any way in which your strategy relates to the grammar?

Now keep the parsing strategy fixed and see whether changing the order of the grammar rules allows you to get a successful parse.

To do this, you’ll have to download a bug-corrected version of srparser-app.py which we’ve mounted, and run python on it directly in order to launch the demo.

```python
>>> python srparser-app.py
```

Once you do this, you can use the Edit menu at the top of the display window to edit the grammar. Since “cut and paste” doesn’t appear to work, to change the order of a rule, you’ll have to delete it, and then type it in where you want it to be. When you’re finished, hit the “OK” button at the bottom, and step through the parser with your different rule ordering. For example, see what (if anything) happens when the rules for VP are reordered, so that the VP rule with the longest RHS is after the others or before the others.

The last thing to do is see what happens when left-recursive rules are removed from the grammar. What happens now with an eager reduction strategy?

2 CYK Chart Parsing

Now we turn to chart parsing. With the CYK chart parser, it’s important to understand both what gets put in the chart and in what order. The order is important, as a guarantee that no possible constituent is ignored.
We’ll use the simple grammar:

```
S → NP VP  DET → a | the (determiner)
NP → DET N  N → joiner | saw (noun)
NP → N  POSS → 's
DET → NP POSS  V → saw (verb)
NP → PRO  PRO → I (pronoun)
VP → V NP
```

2. We’ll start with the sentence

```
I saw the joiner’s saw.
```

which we’ll consider to be already tokenized and indexed as

```
0 I 1 saw 2 the 3 joiner 4 ’s 5 saw 6
```

Using the pseudo-code from Lecture 16 (repeated below) and the matrix at the end of the tutorial, simulate the CYK chart parser on this sentence, numbering the entries in the chart in the order in which you’ve recorded them.

Let Close(X) = \{ B | B →* A, using unary productions, and A ∈ X \}

```
BUILD_CYK_CHART(t, [w_1, \ldots, w_n])
for j ← 1 to n
do
  t(j-1, j) ← Close({w_j})
for k ← 1 to n
for j ← k to n
for m ← 1 to k-1
do
  t(j-k, j) ← t(j-k, j) ∪ Close( \{ A | A → B C for some B ∈ t(j-k, j-m) and C ∈ t(j-m, j) \})
```

3. How many complete analyses of S did CYK find for this sentence or any of its substrings, and what positions do they span?

### 3 Active Chart Parsing

Here we’ll review active chart parsing, using the same grammar as we used with CYK parsing and the same tokenized and indexed sentence

```
0 I 1 saw 2 the 3 joiner 4 ’s 5 saw 6
```

4. Give the dotted rules that can be derived from NP. If you can do this correctly, there’s no need for you to give dotted rules for the other non-terminals, just as long as you remember to use them.

When you answer Questions 5 and 6 below, characterize the arcs that would be added to the chart in terms of the rule used, the starting position of the arc and its ending position.

5. Assuming we’re at the very start of the process

a. What would the Top-Down Initialization Rule add to the chart at the very start of the process?

b. What would the Bottom-Up Initialization Rule add to the chart at the very start of the process?

c. What would the Earley algorithm Predictor Rule add to the chart at the very start of the process? (Remember that the Earley algorithm does not predict words, only parts-of-speech. Its Scanner Rule links parts-of-speech with words in the input.)
6. Assuming the chart already contains the edges

- \( S \rightarrow \ NP \ VP \ (0, 0) \)
- \( NP \rightarrow \ DET \ N \ (0, 0) \)
- \( DET \rightarrow \ NP \ POSS \ (0, 0) \)
- \( NP \rightarrow \ N \ (0, 0) \)
- \( NP \rightarrow \ PRO \ (0, 0) \)
- \( I \rightarrow \ (0, 1) \)

a. What arc or arcs would all possible applications of the *Bottom-Up Predict Rule* add to the chart?

b. Following this, what arc or arcs would all possible applications of the *Fundamental Rule* (aka Earley’s *Completer Rule*) then add to the chart?

c. Following this, what arc or arcs would all possible applications of the *Top-Down Expand Rule* then add to the chart, starting at position 1?

You can use NLTK’s demo on chart parsing

```python
>>> from nltk.app import chartparser
>>> chartparser()
```

to develop a sense of what these and other rules will add to the chart under different circumstances.

4 Probabilistic CFGs (PCFG)

Consider the following tiny PCFG

<table>
<thead>
<tr>
<th>Rule</th>
<th>Production</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>( S \rightarrow \ NP \ VP )</td>
<td>(1.0)</td>
</tr>
<tr>
<td>R2</td>
<td>( NP \rightarrow \ DET \ N )</td>
<td>(0.7)</td>
</tr>
<tr>
<td>R3</td>
<td>( NP \rightarrow \ NPR )</td>
<td>(0.3)</td>
</tr>
<tr>
<td>R4</td>
<td>( VP \rightarrow \ V \ PP )</td>
<td>(0.7)</td>
</tr>
<tr>
<td>R5</td>
<td>( VP \rightarrow \ V \ NP )</td>
<td>(0.3)</td>
</tr>
<tr>
<td>R6</td>
<td>( PP \rightarrow \ Prep \ NP )</td>
<td>(1.0)</td>
</tr>
<tr>
<td>R7</td>
<td>( NPR \rightarrow \ John )</td>
<td>(0.5)</td>
</tr>
<tr>
<td>R8</td>
<td>( NPR \rightarrow \ Mary )</td>
<td>(0.5)</td>
</tr>
<tr>
<td>R9</td>
<td>( V \rightarrow \ saw )</td>
<td>(0.4)</td>
</tr>
<tr>
<td>R10</td>
<td>( V \rightarrow \ loves )</td>
<td>(0.6)</td>
</tr>
<tr>
<td>R11</td>
<td>( DET \rightarrow \ a )</td>
<td>(1.0)</td>
</tr>
<tr>
<td>R12</td>
<td>( N \rightarrow \ cat )</td>
<td>(0.6)</td>
</tr>
<tr>
<td>R13</td>
<td>( N \rightarrow \ saw )</td>
<td>(0.4)</td>
</tr>
</tbody>
</table>

7. With respect to this grammar, what is the probability of the sentence

\[ \text{John saw a saw.} \]

8. If a fire alarm starting ringing, so that you only caught the first two words of a sentence which started

\[ \text{John saw . . .} \]

with respect to this grammar, what is the most likely completion of the sentence?
I saw the joiner's saw.