LING185A, Assignment #6

Due date: Wed. 2/21/2018

Background and assumptions

A sample grammar

Here's a simple CFG you should use for most of the questions below.

S → NP VP	N → dog, cat, rat, wife, brother
S → WHILE S S	NP → John, Mary
NP → NP POSS N	V → barked, chased, bit, ate, fled
NP → (D) N (PP) (SRC) (ORC)	D → the
VP → V (NP) (PP)	P → on, in, with
PP → P NP	THAT → that
SRC → THAT VP	POSS → 's
ORC → NP V	WHILE → while

Review of left-corner parsing

Recall that left-corner parsing can be thought of as a combination of bottom-up and top-down parsing. The stack of a left-corner parser mixes (a) symbols with a bar over them, which represent still-to-come constituents (tree nodes with no children yet), and are analogous to the symbols on the stack in top-down parsing, and (b) symbols without a bar over them, which represent already-recognized constituents (tree nodes with no parent yet), and are analogous to the symbols on the stack in bottom-up parsing. The *left-corner* of a context-free rule is the first symbol on the right hand side.

To illustrate, here's the example we saw in class.

	Type of step	Rule used	Configuration
0	—	—	($\bar{\text{S}}$, the dog chased the cat)
1	SHIFT	D → the	(D $\bar{\text{S}}$, dog chased the cat)
2	LC-PREDICT	NP → D N	($\bar{\text{N}}$ NP $\bar{\text{S}}$, dog chased the cat)
3	MATCH	N → dog	(NP $\bar{\text{S}}$, chased the cat)
4	LC-CONNECT	S → NP VP	($\overline{\text{VP}}$, chased the cat)
5	SHIFT	V → chased	(V $\overline{\text{VP}}$, the cat)
6	LC-CONNECT	VP → V NP	($\overline{\text{NP}}$, the cat)
7	SHIFT	D → the	(D $\overline{\text{NP}}$, cat)
8	LC-CONNECT	NP → D N	($\bar{\text{N}}$, cat)
9	MATCH	N → cat	(ϵ, ϵ)

The SHIFT and MATCH rules should be relatively easy to understand. They are similar to the rules of the same names in bottom-up and top-down parsing. Notice that the SHIFT rule, since it does bottom-up work, only manipulates "plain" symbols (e.g. D at step 1, V at step 5), and the MATCH rule, since it does top-down work, only manipulates "barred" symbols (e.g. $\bar{\text{N}}$ at step 3).

The LC-PREDICT rule is a bit more complicated. It manipulates both plain and barred symbols. The application of LC-PREDICT at step 2, for example, transitions from ‘D $\bar{\text{S}}$’ to ‘$\bar{\text{N}}$ NP $\bar{\text{S}}$’. This step is based on the fact that D is the left-corner of the rule ‘NP $\rightarrow$ D N’: having recognized (in a bottom-up manner) a D, we predict (in a top-down manner) an N; and if we manage to fulfill that prediction of an N, we will have recognized (in a bottom-up manner) an NP.

The LC-CONNECT rule, like the LC-PREDICT, predicts “the rest” of the right-hand side of a rule on the basis of having already recognized the rule’s left-corner. For example, when we’ve reached ‘NP $\bar{\text{S}}$’, the application of LC-CONNECT at step 4 is based on the fact that NP is the left-corner of the rule ‘S $\rightarrow$ NP VP’. The difference here, compared to the situation above with LC-PREDICT, is that we already have a *predicted* S (i.e. the symbol $\bar{\text{S}}$) waiting for us there on the stack. Since we’ve already found an NP bottom-up, the prediction of an S will be fulfilled if we find a VP in the coming words, so it can be swapped for a predicted VP (i.e. the symbol $\overline{\text{VP}}$).

1 Stack depth and different embedding structures

Recall the three kinds of embedding structures we’ve considered. The asterisk on (3c) just records the empirical fact that English speakers say “yuck!” in response to this sentence (what we might call *unacceptability*).

(1) Left-branching structures
 a. John ’s dog barked
 b. John ’s brother ’s dog barked
 c. John ’s brother ’s wife ’s dog barked

(2) Right-branching structures
 a. Mary chased the cat
 b. Mary chased the cat that bit the rat
 c. Mary chased the cat that bit the rat that ate the cheese

(3) Center-embedding structures
 a. the rat fled
 b. the rat the cat chased fled
 c. * the rat the cat the dog bit chased fled

We discovered in class that when the bottom-up parsing schema is applied to left-branching structures, the required memory load (i.e. the largest stack size that is required) *does not* increase when the pattern shown in (1) is extended to longer and longer sentences. With right-branching and center-embedding structures, on the other hand, the memory load on a bottom-up parser *does* increase as these patterns are extended. We also saw that on right-branching structures, the top-down parser behaves differently: the required memory load for a top-down parser *does not* increase as the pattern in (2) is extended to longer and longer sentences. But with the top-down parser, alas, the memory load does increase when the pattern in (1) is extended.

We can summarize these findings, and compare them with the facts about humans that we’d like to account for, in the table below. Each cell describes what happens with long sentences of a particular sort *compared to* what happens with short sentences of the same sort. So in the row describing the bottom-up parser, ‘no increase’ in the left-branching column indicates that (1c) *does not* require any more stack space than (1b) requires, whereas ‘increase’ in the right-branching column indicates that (2c) *does* require more stack space than (2b) requires.

	left-branching	right-branching	center-embedding
Human difficulty	no increase	no increase	increase
Bottom-up parser	no increase	increase	increase
Top-down parser	increase	no increase	
Left-corner parser			

Your task here — in case you haven't already guessed — is to fill in the remaining four cells of the table. Use the grammar in given above.

So for each of the four cells, you need to show what happens in the relevant 'b.' sentence and the relevant 'c.' sentence. For the 'b.' sentences, show the *full* sequence of configurations from start to goal; use a format something like that of the table on page 1. For the 'c.' sentences, you can skip some steps, but make sure that you show the configurations that impose the largest memory load. For all eight parses, indicate clearly what the largest required memory load (i.e. maximum number of symbols held on the stack) is.

2 More on stack depth

Let's suppose that there is a certain kind of Martians that, to the best of our knowledge, have the grammar given above as their mental grammar. Then we discover certain new kinds of sentences that Martians judge to be acceptable, that have not been noticed before. Here are some examples of this new kind of sentence:

(4) a. John said loudly Mary fled.
b. Mary said quietly John 's wife ate.
c. the dog said slowly the rat bit Mary.

So now the question is, what new rules should we add to the grammar above in order to account for these new kinds of sentences? We will consider two hypotheses.

- **Hypothesis #1:**
  VP → SAID ADV S
  SAID → said
  ADV → loudly, quietly, slowly
- **Hypothesis #2:**
  VP → X S
  X → SAID ADV
  SAID → said
  ADV → loudly, quietly, slowly

Note that no matter which of these two hypotheses we adopt, the new grammar will generate exactly the same set of sentences. So there is no way to distinguish between these two hypotheses on the basis of which sentences they classify as grammatical or ungrammatical.

We do know, however, that these Martians use bottom-up parsing. And it's generally accepted that a memory limitation is responsible for the fact that these Martians judge (5a) to be perfectly acceptable, but give a "yuck!" reaction (as indicated by the asterisk) to (5b). Of course (5b), like (5a), is generated by the rules in the our current grammar and is therefore assumed to be grammatical for these Martians (just like (3c) above is for speakers of English).

(5) a. Mary chased the cat
b. * Mary chased the cat that bit the rat

In order to decide between Hypothesis #1 and Hypothesis #2, a clever linguist devises the following sentences and asks Martians for their judgements of them. As above, the presence or absence of the asterisk indicates the presence or absence of a "yuck!" reaction.[1]

(6)
- a. John ate
- b. Mary said quietly John ate
- c. John said slowly Mary said quietly John ate
- d. * John said slowly John said loudly Mary said quietly John ate

How can we use this information to choose between Hypothesis #1 and Hypothesis #2? Explain your reasoning fully.

3 Garden paths and reanalysis

Sentences like the one in (7) tend to produce a "garden path" effect (at least temporarily):

(7) While John ate the dog that barked fled.

On the first pass through this sequence of words, people tend to interpret 'the dog' as the beginning of an NP that is the object of 'ate'. [2]

For the following questions, use the grammar given at the beginning of this assignment.

A. Draw the complete correct parse tree for (7). Also draw a "partial tree" that shows the structure that the human processor seems to initially assign to the words up and including 'barked'.

B. Show the full sequence of configurations that a bottom-up parser goes through to *correctly* parse the sentence in (7). Identify the point in this correct sequence of configurations where humans (if they were using a bottom-up parsing system) initially head off in the wrong direction through the search space; show at least the first three "wrong" configurations that it reaches.

C. Do the same thing, now for a top-down parser.

D. Let's suppose that when the human parsing system "backtracks" (i.e. reaches a dead end and has to go back a few steps to start off down a different path), this works simply by looking back (right-to-left) through the sentence in an obvious "dumb" way (i.e. going back to the point in the sentence where it took a wrong turn and trying again). Explain how we could use an experiment where we track the movements of people's eyes as they read through (7) to decide whether humans use a bottom-up parser or a top-down parser. (Assume for this question that these are the only two possibilities.)

4 FSAs and finite memory, CFGs and unbounded memory

The aim of this question is to illustrate the relationship between *states* in an FSA and *sequences of nonterminal symbols* in a CFG; and in turn, to illustrate the sense in which a CFG is an "infinite state machine".

[1] The clever linguist did not bother to find out whether, in the sentences where there are multiple occurrences of 'John', the Martians take these occurrences to necessarily refer to distinct people all called John, as would be the case for many humans. In fact, the clever linguist didn't bother to find out anything at all about how Martians interpret these sentences — it's currently a topic of debate whether the Martian word 'John' is a name at all, or whether it's an adjective meaning something like "pertaining to small quantities of brake fluid" — because her goal was just to decide between Hypothesis #1 and Hypothesis #2, and she couldn't see a way to use information about interpretations to help make that decision.

[2] This is an example of the "Late Closure" parsing preference: the structure people go to first is one where the VP constituent is "closed off" later in the sentence than the first possibility, which is straight after 'ate'. If you don't get the effect, try something like 'Although John kept reading the story about the dog made him sad'.

Consider the stringset $L = \{\mathrm{a}^n\,\mathrm{c}\,\mathrm{b}^n : n \geq 0\} = \{\mathrm{c}, \mathrm{acb}, \mathrm{aacbb}, \mathrm{aaacbbb}, \dots\}$, i.e. the set of strings made up of some number of 'a's, then a single 'c', and then the same number of 'b's. As we have seen, there is no FSA that can generate this stringset; but it can generated by the following very simple CFG:

(8)
$\mathrm{S} \rightarrow \mathrm{A\ S\ B}$
$\mathrm{S} \rightarrow \mathrm{C}$
$\mathrm{A} \rightarrow \mathrm{a}$
$\mathrm{B} \rightarrow \mathrm{b}$
$\mathrm{C} \rightarrow \mathrm{c}$

Someone less knowledgeable about these things than you, however, who was still wondering whether it might be possible to design an FSA that generates L, might be tempted to start by writing out something like this:

(9)

$$\begin{tikzcd}
\text{START} \arrow[r] & \text{0A} \arrow[r, "a"] \arrow[d, "c"] & \text{1A} \arrow[r, "a"] \arrow[d, "c"] & \text{2A} \arrow[r, "a"] \arrow[d, "c"] & \text{3A} \arrow[r, "a"] \arrow[d, "c"] & \text{4A} \arrow[d, "c"] \\
 & \text{0B} & \text{1B} \arrow[l, "b"'] & \text{2B} \arrow[l, "b"'] & \text{3B} \arrow[l, "b"'] & \text{4B} \arrow[l, "b"']
\end{tikzcd}$$

This un-knowledgeable person has labeled the states in a way that helps us understand what it "means" to be in each state: for example, being in state 3A means that you've produced three 'a's, being in state 2B means that you need to produce two 'b's to reach the final state, etc. But remember that these labels are just to make things a bit more comprehensible to humans: as far as the machine is concerned, they are just ten states with ten different names, and they might as well just be labeled with the letters A through J (or the letters M through V, or whatever).

Of course, the FSA in (9) does not actually generate L: it doesn't generate any strings longer than 'aaaacbbbb'. But by working through the following exercises you should gain a sharper appreciation of exactly what it is that the CFG in (8) is doing that the FSA in (9) is not (and that no FSA can).

A. Show the sequence of configurations that a parser would go through in parsing the string 'aaaacbbbb' using the FSA in (9), using the FSA parsing schema from the class handout.

B. Show the sequence of configurations that a top-down parser would go through in parsing the string 'aaaacbbbb' using the CFG in (8).

C. Show the sequence of configurations that a bottom-up parser would go through in parsing the string 'aaaacbbbb' using the CFG in (8).

D. Show the sequence of configurations that a left-corner parser would go through in parsing the string 'aaaacbbbb' using the CFG in (8).