Assignment #7

Due date: Thu. 2/25/2021, 12 noon

Background and assumptions

A sample grammar

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

S → NP VP	N → baby, boy, actor, award, boss
S → WHILE S S	NP → Mary, John
NP → NP POSS N	V → met, saw, won, cried, watched
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 transition	Rule used	Configuration
0	—	—	($\overline{\text{S}}$, the baby saw the boy)
1	SHIFT	D → the	(D $\overline{\text{S}}$, baby saw the boy)
2	LC-PREDICT	NP → D N	($\overline{\text{N}}$ NP $\overline{\text{S}}$, baby saw the boy)
3	MATCH	N → baby	(NP $\overline{\text{S}}$, saw the boy)
4	LC-CONNECT	S → NP VP	($\overline{\text{VP}}$, saw the boy)
5	SHIFT	V → saw	(V $\overline{\text{VP}}$, the boy)
6	LC-CONNECT	VP → V NP	($\overline{\text{NP}}$, the boy)
7	SHIFT	D → the	(D $\overline{\text{NP}}$, boy)
8	LC-CONNECT	NP → D N	($\overline{\text{N}}$, boy)
9	MATCH	N → boy	(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. $\overline{\text{N}}$ 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 → 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 → 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 $\overline{\text{VP}}$).

One catch to look out for is that when LC-PREDICT or LC-CONNECT applies to a unary grammar rule, there are no new barred symbols created, because the only thing on the right-hand side of the rule *is* the left-corner. For example, the last step of parsing ‘the baby won’, using the unary rule ‘VP → V’, is one such situation.

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

1 Stack depth and different embedding structures

Recall the three kinds of embedding structures we’ve considered, repeated here. Increasing the depth of center-embedding structures seems to cause a certain kind of processing difficulty in humans, which does not arise when we increase the depth of left-branching or right-branching structures.

(1) Left-branching structures
  a. Mary won
  b. Mary ’s baby won
  c. Mary ’s boss ’s baby won

(2) Right-branching structures
  a. John met the boy
  b. John met the boy that saw the actor
  c. John met the boy that saw the actor that won the award

(3) Center-embedding structures
  a. the actor won
  b. the actor the boy met won
  c. * the actor the boy the baby saw met won

I’ve put an asterisk on (3c) just to indicate that this sentence produces a “yuck, something went wrong” reaction from English speakers. Logically speaking, this *could* be because the sentence is *ungrammatical*, i.e. not generated by English speakers’ mental grammars; but based on the other sentences in (3), a more attractive hypothesis might be that there is *something else* going wrong in (3c), which is producing the “yuck” reaction that we are recording with the asterisk. (When we want to be careful about this kind of thing, we might say that the asterisk on (3c) indicates only that it is *unacceptable*, which may or may not be because it is *ungrammatical*.[1] Unfortunately there is some inconsistency, in general, about whether asterisks are indicators of unacceptability or ungrammaticality, because the large majority of the time they are taken to coincide.)

So the important generalization is this: considering larger and larger center-embedding structures eventually leads to sentences that produce a "something went wrong" reaction, whereas we do not see this for left-branching or right-branching structures.

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 (1)	right-branching (2)	center-embedding (3)
Human difficulty	no increase	no increase	increase
Bottom-up parser	no increase	increase	increase
Top-down parser	increase	no increase	
Left-corner parser			

A. Your task here — in case you haven't already guessed — is to fill in the remaining four cells of the table. Use the grammar given above and the sentences in (1)–(3). So for each cell, you need to consider 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 the format of the table on page 1, *showing the transition type and rule used for each step*. For the 'c.' sentences, you don't need to show all the gory details, but for full credit you must at least write down the configurations that impose the largest memory load and clearly state how that load compares with the corresponding 'b.' sentence; providing a brief explanation of your thinking in addition might allow us to assign partial credit if any of the specifics are wrong.

B. The left-corner parsing system that we've introduced so far is, more specifically, known as *arc-eager left-corner parsing*. Another variant is known as *arc-standard left-corner parsing*. In this alternative system, the SHIFT, MATCH and LC-PREDICT rules are unchanged, but there is no LC-CONNECT rule; instead, there is a different fourth rule, which we can call CANCEL, which is defined like this:

CANCEL step: $(A\overline{A}\mathbf{\Phi}, x_i \dots x_n) \Longrightarrow (\mathbf{\Phi}, x_i \dots x_n)$
where A is any nonterminal in the grammar

How does this arc-standard variant of left-corner parsing differ from the arc-eager version that you used above, regarding predictions for left-branching, right-branching and center-embedding structures? For

[1] Besides (3c), some other examples of sentences that are unacceptable for many English speakers but probably grammatical are 'Police police police police police police' and 'The horse raced past the barn fell'; although they definitely produce a "something wrong" reaction, our guess is that this is best attributed to non-grammatical factors. Some pretty mysterious examples of sentences that are acceptable for many English speakers but probably ungrammatical are 'The actor the boy the baby saw won' and 'More people have been to Russia than I have'. And there are always people pursuing the idea that sentences like 'Who do you wonder whether John met yesterday?', while certainly unacceptable, are in fact grammatical — although people developing the kind of theories we teach in linguistics classes here generally take these to be ungrammatical.

full credit you must at least (i) show the complete sequence of configurations that an arc-standard left-corner parser goes through to process sentence (3b), using the format of the table on page 1 as above, and (ii) write down the configurations that impose the largest memory load for all three 'b.' sentences and all three 'c.' sentences.

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 these Martians produce and/or judge to be acceptable sentences of their language, that have not been noticed before. Here are some examples of this new kind of sentence:

(4) a. John said loudly Mary won
b. Mary said quietly John 's boss won
c. the boss said slowly the actor met 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 the two grammars generate.*

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, although these Martians find (5a) to be a perfectly unremarkable sentence, (5b) makes their heads spin.[2] So although (5b), like (5a), is generated by the rules in our current grammar and is therefore assumed to in fact be *grammatical* for these Martians, there is something else going wrong when they try to process (5b). To record this observed difference I'll put an asterisk next to those sentences that produce a "something-went-wrong" reaction from the Martians.

(5) a. John met the boy
b. * John met the boy that saw the actor

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 something-went-wrong reaction.[3]

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

[2] Not literally. There are other sentences which make that happen but we can leave those aside for the purposes of this question.

How can we use this information to choose between Hypothesis #1 and Hypothesis #2? Explain your reasoning fully. Notice that in this question, unlike above, the issue is not whether or not considering larger and larger structures with a certain kind of embedding pattern leads to processing difficulty *at some point* — it clearly does — but rather *at which point* we see the difficulty arising.

3 Garden paths and reanalysis

Sentences like the one in (7) tend to produce a "garden path" effect[4] (at least temporarily):

(7) While John watched the baby that won cried

On the first pass through this sequence of words, people tend to interpret 'the baby' as the beginning of an NP that is the object of 'watched'.[5]

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 'won'.

B. Show the full sequence of configurations that a bottom-up parser goes through to *correctly* parse the sentence in (7). (Including the labels of the transitions and grammar rules used is optional here.) Identify the point in this correct sequence of configurations where humans (assuming for the moment that they use 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 configuration 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 way (i.e. going back to the point in the sentence where it took a wrong turn, and trying again); and suppose that this backtracking can be detected in experiments where we track the (leftward and rightward) movements of people's eyes as they read. Explain how we could use (7) in such an experiment 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".

Consider the stringset $L = \{\mathrm{a}^n\,\mathrm{c}\,\mathrm{b}^n \mid n \geq 0\} = \{\mathrm{c}, \mathrm{acb}, \mathrm{aacbb}, \mathrm{aaacbbb}, \dots\}$, 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) S → A S B
S → c
A → a
B → b

[3]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.

[4]`https://en.wikipedia.org/wiki/Garden-path_sentence`

[5]More specifically, this is an example of what's known as 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 'watched'. If you don't get the effect, try something like 'Although John kept reading the story about the baby made him sad'.

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)

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

Of course, the FSA in (9) does not actually generate L: it doesn't generate any strings longer than 'aaacbbb'. 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 'aaacbbb' using the FSA in (9), using the FSA parsing schema at the beginning of the class handout.

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

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

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

(And, with a little bit of further thought, you might notice a connection between (i) the CFG/FSA relationship as illustrated by this last section, and (ii) the fact that left-branching and right-branching structures can be parsed without making use of the unbounded potential capacity of the stack, but center-embedded structures cannot. After all, with left-branching and right-branching structures, the inside values are really just forward and backward values — only prefixes and suffixes get put into equivalence classes!)