Tutorial 4
Shift-Reduce Parsing

1. Review the Shift-Reduce algorithm carefully in the lecture slides or in *Mechanisms for Sentence Processing*, Crocker, 1999.
2. Make a copy of Parser Architecture. Review the model, taking the time to check refractedness of rules and initialization times of buffers and processes.
3. Modify it to include our latest architectural changes:
   a) Replace the trigger in the sentence selection rule with a condition which will allow the rule to fire only on cycle 1.
   b) Add the mechanism which prevents words from being "read" over and over again.
4. Run once through both stimuli to make sure the architecture works correctly.
5. Create a copy of the model and name it Shift-Reduce.
6. Then start going through the same development steps as for the top-down parser by removing the stub in Parse and implementing the shift-reduce parse rules as given in the lecture slides. Note in particular the order of the daughters of a rule on the stack!
7. Debug this basic model, using grammar rules of the form `rule(s, np, vp)` and the two sentences *"John loves Mary"* and *"The cat chases the mouse"*.
8. In your stimuli, change *"John loves Mary"* to *"John loves Mary tenderly"*. Then add `vp → vp adv` to your grammar (this rule is left-recursive, but this is not a problem for the shift-reduce parser) and the adverb to your lexicon.
9. Draw the search tree for the new sentence, then try to parse it with your model, and add cycle numbers next to each node in your search tree.
10. You will notice that this sentence cannot be parsed correctly. The problem occurs after cycle 13, when [np vp] is reduced to [s] before the adverb is shifted, so that vp → vp adv cannot be applied, and we are finally left to fail with [s adv] on the

stack. Instead, after cycle 13, we want the system to become quiescent, clear the display and then show *tenderly*. This means we have to prevent `reduce` from applying before *tenderly* has arrived in Current Word.

11. If you now try to reparse the sentence, you will notice that all choice points are now correctly detected, but that this sometimes leads to more than one element on the stack. This means that since we want a serial parser, we are going to have to add a "propose-execute" cycle and rule selection as we did in the top-down model. Split your rules into two parts and add the random selection mechanism from section 7.2 in the the top-down tutorial.

12. Debug your rules by parsing the beginning of the sentence until cycle 21.

13. Once your rules work, re-parse the sentence until the parser makes the "right" choices at the choice points, and see if you can parse the sentence all the way to the end. You should notice that the parser fails at cycle 24, even when it is on the right track. Make the necessary change so that the sentence can be parsed until the end (Hint: This involves modifying your stimuli).

14. Now, modify your parse rules to accept n-ary grammar rules:
    a) Make *John* and *Mary* proper nouns in the lexicon, add the adjective,
    b) Add `np` → `pn` and `np` → `det adj n` to the grammar,
    c) Comment out unnecessary rules in the grammar to accelerate parsing,
    d) Parse *"John chases the black cat"*.

    This is a little more difficult than adding n-ary rules to the top-down parser because of the ordering of the daughters on the stack. You will need to use append and reverse. Consider creating an archive or making a copy of your model before making this modification, as it appears COGENT may contain bugs when using append and reverse which could render your model unusable.

15. Finally, add backtracking to the model. Remember you will need to make small changes to the success and fail rules to adapt them to shift-reduce, too.