Parsing in Prolog

Tutorial 1

Name:

The Parser Files

- The source code for the parsers we talked about in the lecture can be found at `/proj/courses/comppsych/Tutorial1`
- Copy the files to your own directory.

Shift-Reduce Parser

1. **How does the parser work?**
   Load `shift-reduce.pl` into the Prolog system. There are four pre-defined sentences called `easy, centre, right` and `left`.

   A. To parse any of these sentences, type `do(SentenceName).` What happens if you type literally this, and why?

   B. At each point during sentence processing, the parser outputs the stack (newest item first) and the list of words to be processed. Parse the `easy` sentence.
   What happens? Is this a top-down or a bottom-up parser?

   C. Remember, a shift step reads a new word and a reduce step applies a grammar rule to elements on the stack to replace them with a new category. Name the steps the parser goes through, also marking if backtracking happens.

   D. Parse the `centre, left` and `right` sentences and look at the way the stack behaves. Do you see differences? Describe them.

2. How is this implemented?

- Open `shift-reduce.pl` in your favourite editor. The most interesting bits for now are the rule definitions for parsing and the grammar and lexicon definitions.
- Look at the grammar first. It's a straightforward CFG that accounts for sentences with relative clauses in them.
- Now let's look at the actual parsing. This is done by a Prolog predicate called `parse`. It takes as its arguments the stack and a list of words that still need to be incorporated. There are three possible ways of proving that `parse` is true: Either, the stack only contains an S and there are no more words in the list. We're done! All that needs to be done is to write the empty list. Otherwise, we can reduce (apply a grammar rule) or shift (read a new word).

A. Give the variable bindings for unifying

```
parse([np], [read, the, book]) and
parse(Stack, [Word|Rest]) :- (Cat ---> [Word]),
parse([Cat|Stack],Rest).
```

Stack: Word: Rest: Cat:

B. Try the following query. Use the `;` after each match to see if there are more. What do you get?

```
?- X ---> RHS.
```

C. Have a close look at the way the shift and reduce rules work and explain in your own words what happens:

D. You're now ready to have a closer look at the way Prolog parses. At the Prolog prompt, type `trace`. This will make the interpreter slow down and show you every step of its calculations. It starts getting interesting when `parse` is first called. Look at the output and try to find out which rules are being called. Can you explain why the first two calls must be shifts? If you are getting really annoyed with the written output, you can comment out the lines that do it (the comment symbol is `%`).
Can you find the place where things start to go wrong and Prolog has to backtrack?