Testing the Parser: Context Free Language Generation

The possibility exists of writing a program to develop test inputs for the parser. The design of such a program indicates how we can think about these tests.

The term *language* in our context means a set of *words* that are comprised of only terminal symbols. However, there are lots of intermediate forms derived during a parse. Since we're using a recursive descent parser with a programming discipline of writing a recursive function for each categorical symbol, the appearance of a categorical symbol within the words indicates where programs are called. For example, a full parse of

[[+ 1 2]]

using the project grammar looks like this.

T => [S] => [expr] => [oper] => [[binops oper oper]] => [[+ oper oper]] => [[+ constants oper]] => [[+ ints oper]] => [[+ 1 oper]] => [[+ 1 constants]] => [[+ 1 ints]] => [[+ 1 2]]

Looking at this, we can see that, while there might be an infinite number of ways to write the last (terminal) word, there is only one **form:** namely, *(atom atom atom).* This is the key to developing tests.

For formality sake, we can define the *length of a word* as the number of (terminal or categorical) symbols in the word. *[+ 1 2]* has length 5. The key to testing, then, is to think of various ways you can generate words that are only comprised by terminal symbols in such a way that all the possible categorical symbols are also involved.

Assignment 1

What is the shortest length terminal word? Show its derivation. What is the next shortest? Show its/their derivation.

Assignment 2

Demonstrate a terminal word of length *n=1, ..., 5* or show that it is impossible to have a word of that particular length.

Assignment 3

Outline how you could rewrite the parsing algorithm to a program that can generate the legal strings of any length.