

CS 405/505 ASSIGNMENT #6

Due Thursday, March 10, 2005

1. Write a Prolog predicate `derive(P, S, L, T)`, where `P` is a list of context-free grammar production rules of the form `rule(A, [X1, X2, ..., Xn])` corresponding to $A \rightarrow X_1X_2\dots X_n$, `S` is the start symbol of the grammar, and `L` is a list of integers, and `T` is the parse tree resulting from applying the productions in `P` in the order indicated by `L`. The first production rule to be applied must be a production of the start symbol or a null tree is returned. The process terminates if the integer indicated does not match any production rule which can be applied, e.g. is either not a production rule or there is no corresponding nonterminal. If there are more than one nonterminal to apply a production rule to, the leftmost one is always selected. The parse tree will be of the form `parsetree(A, [X1, X2, ..., Xn])` for $A \rightarrow X_1X_2\dots X_n$, with each `Xi` replaced by its corresponding parse tree. Test your function on the following:

```
derive([rule(e, [e, +, t]), rule(e, [t]), rule(t, [t, *, f]), rule(t, [f]),
       rule(f, ['(', e, ')'])], rule(f, [id]), e, [1, 2, 4, 6, 3, 4, 6, 6], ParseTree)
⇒ ParseTree = parsetree(e, [parsetree(e, [parsetree(t, [parsetree(f, [id])])]),
                             +, parsetree(t, [parsetree(t, [parsetree(f, [id])])]), *, parsetree(f, [id])])])
```

```
derive([rule(e, [e, +, t]), rule(e, [t]), rule(t, [t, *, f]), rule(t, [f]),
       rule(f, ['(', e, ')'])], rule(f, [id]), e, [1, 3, 6, 4, 6, 2, 4, 6], ParseTree)
⇒ ParseTree = parsetree(e, [parsetree(e, [parsetree(t, [parsetree(f, [id])])]),
                             +, parsetree(t, [parsetree(t, [parsetree(f, [id])])]), *, parsetree(f, [id])])])
```

```
derive([rule(e, [e, +, e]), rule(e, [e, *, e]), rule(e, ['(', e, ')']),
       rule(e, [id])], e, [1, 4, 2, 4, 4])
⇒ ParseTree = parsetree(e, [parsetree(e, [id]), +,
                             parsetree(e, [parsetree(e, [id]), *, parsetree(e, [id])])])])
```

2. Write a Prolog predicate `parsetrees(P, S, N, T)`, where `P` and `S` are as in the `derive` predicate, `N` is an integer, and `T` is a list of all parse trees reached after `N` or less production rules have been applied. If $N \leq 0$, `T` will be a null list. One way to design this predicate is to produce a set of lists `[0, 0, ..., 0]` (`n` 0's) ... `[m, m, ..., m]` (`n` `m`'s) in order and call `derive` for each one, making a list of the non-null parse trees returned. Note that this approach has complexity m^n , which is exponential and will not be practical for large values of `n`, although it should not cause problems for the test cases below. You are welcome to develop a more efficient algorithm which does not need to try all of the m^n combinations. Test your function on the following:

```
parsetrees([rule(e, [e, +, t]), rule(e, [t]), rule(t, [t, *, f]), rule(t, [f]),
           rule(f, ['(', e, ')'])], rule(f, [id]), e, 8)
parsetrees([rule(e, [e, +, e]), rule(e, [e, *, e]), rule(e, ['(', e, ')']),
           rule(e, [id])], e, 8)
```

3. Consider the Core program segment S below:

```
u := 1; v := 0;
while (v < y) loop
  u := u * x;
  v := v + 1;
end loop;
```

Using the axiomatic semantics of Core, show that the assertion
 $\{y \geq 0\} S \{u = x^y\}$

is correct.

Hint: Use the loop invariant $u = x^v \ \& \ v \geq 0$. The consequence rules may be used to force the other rules into a form consistent with this loop invariant. (You should analyze the program to understand why this loop invariant was chosen.)