CS 164 Programming Languages and Compilers	Spring 2019

Discussion Week of 2/11: Top-Down Parsers

Instructor: Paul N. Hilfinger GSIs: Nikhil Athreya, Vivant Sakore

1. **Derivations**.

   If a string belongs to a particular language, we can find a *parse tree* for it. A single nonterminal roots the parse tree (e.g '`prog`' or '`expr`'). This top-level nonterminal branches off into constituent nonterminals, eventually breaking down into terminals and ϵ-symbols at the leaves. *Top-down parsing* is a method of matching strings via an abstract preorder traversal of a parse tree. We've already seen an example of this: recursive descent! As a recap from lecture:

   - *Leftmost derivation*: a sequence of rules following a preorder traversal of the parse tree.
   - *Rightmost derivation*: see previous definition, but with a right-to-left preorder traversal.

   The *reverse rightmost derivation* is the rightmost derivation in reverse. The reverse rightmost derivation is an instance of bottom-up parsing—the string's lowest level details are recognized first, then mid-level details, finally leading up to the start symbol.

   Consider the following grammar:

   ```
   var  : ID ;
   expr : var
        | '(' 'λ' var '.' expr ')'
        | '(' expr expr ')' ;
   ```

   Use this grammar to construct leftmost and reverse rightmost derivations of the following strings.

   - `(λf.(λx.(f (f x))))`

2. **Recursive Descent Parsers**.

   Try writing a recursive descent parser for the grammar in Problem 1. Assume the existence of `scan(C)`, `next()`, and `ERROR()` as defined in lecture.

3. **Ambiguous Grammars**.

A grammar is *ambiguous* if it permits multiple distinct parse trees for some string. For example, without the order of operations, $12 - 8/4$ could parse as 1 or as 10. Make the following grammar unambiguous, and also give precedence to '/' over '-'.

```
e : INT
  | e ‘-’ e
  | e ‘/’ e ;
```

As another example, consider the following grammar:

```
e : 0 | 1 ;

stmt : e ’;’
     | if e then stmt
     | if e then stmt else stmt ;
```

Give an example of a string in the language that can produce multiple parse trees.

4. **Syntax-directed Translation**.

Parser-generators usually support syntax-directed translation, which is a convenient way to execute an action every time a grammar rule is matched. While defining actions, the variable `$$` refers to a location into which the semantic value of the current symbol can be stored. The variables `$1`, ..., `$n` refer to the semantic values of the symbols used to match the current rule. Here's an example:

```
p : e ’;’          { printf("Result: %d\n", $1); }
e : INT            { $$ = $1; }
  | e ‘-’ e        { $$ = $1 - $3; }
  | e ‘/’ e        { $$ = $1 / $3; } ;
```

Write a syntax-directed translator for the first grammar you wrote for Problem 3. Also, write one for Problem 1.