FILS

Course: Compiler Techniques

Homework #4

Top-down Parsing

I. Consider the following grammar, where `S` is the initial symbol and `{a,b,c,d,e}` is the set of terminal symbols:

S -> F c | A c d | λ
F -> b | A c F e
A -> a

1. Examine the grammar and rewrite it so that an LL(1) predictive parser can be built for the corresponding language.
2. Compute the FIRST and FOLLOW sets for all non-terminal symbols in the new grammar
3. Build the parse table.
4. Show the parser configuration (stack, input, and actions) for the analysis process of the **acbec** input sequence.

II. Consider the following grammar:

A -> A a | A b a | a | A b c A | A b c b | c A | c b

a. Transform it in a top-down parsable grammar.
b. Calculate the needed FIRST, FOLLOW for building SD sets.
c. Build the LL(1) parse table for it.
d. Parse the input string **a b c b a** with the aid of a parsing simulation table as in the followings:

STEP	STACK	INPUT	ACTION
1	$A	abcba$	A->xB
2			

III. Programming exercise:
Given the following LL(1) grammar of arithmetic expressions, write a recursive descent parser for this grammar in Java:

0. Goal –> Expr
1. Expr –> Term Expr'
2. Expr' –> + Term Expr'
3. | - Term Expr'
4. | λ
5. Term –> Factor Term'
6. Term' –> * Factor Term'
7. | / Factor Term'
8. | λ
9. Factor –> (Expr)
10. | num
11. | id

Verify the parser using the following two input strings:

(num +id)*id and **id – num * id/()**

Note. A recursive descent parser is a kind of predictive parser.

Hint.

Given a grammar that has the LL(1) property you can write simple routines to recognize possible structures for each non-terminal. The code for such a routine is both simple and fast:

Consider a LL(1) grammar and all A-productions in this grammar:

A -> β_1 **|** β_2 **|** β_3, with

SD(A-> β_i**)** ∩ **SD (A->** β_j**) = ∅**, with **i, j = 1..3, i ≠ j**

Write for A a method with the following algorithm:

```
/* select a non-terminal A */
public boolean A() {
   if (current_token ∈ SD(A→β1))
         find an input substring β1 and the return true
   else if (current_token ∈ SD(A→β2))
         find an input substring β2 and return true
   else if (current_token ∈ SD(A→β3))
         find an input substring β3 and return true
   report an error and return false
}
```

Add a `main()` method that verifies the parser.