# HW 5 – Top-down parsing

CS 421 – Spring 2013

Revision 1.0

**Assigned** Feb. 12, 2013
**Due** Sunday, Feb. 17, 2013, 9:00 AM
**Extension** No lateness
**Total points** 50

## 1 Change Log

**1.0** Initial Release.

## 2 Overview

This homework will give you practice with the concepts we covered in lecture 9. You will be asked, among other things, to write some programs in OCaml. You do not need to run these.

Note that there is no late submission allowed for this assignment (because we want to release the solutions on Sunday). Keep in mind that there is partial credit; hand in whatever you have done by Sunday morning.

## 3 Handing in the assignment

The handin process is the same as for hw4 (except, obviously, use "`/class/cs421/handin -s hw5`".) You must submit electronically, as a pdf. Parse trees may either be included as pictures or given using indentation.

## 4 Problems

In the first four problems, you are given a grammar. For each grammar, answer the following questions:

(a) Give the FIRST sets of all non-terminals.[1]

[1] Since we have not given an algorithm, you can only calculate FIRST sets "by inspection" - that is, by just doing your best. We will grade this part of the problem liberally, and will not penalize you if an incorrect answer here leads to incorrect answers to the other questions.

(b) Using the results from item 1, give the FIRST sets of all right-hand sides of production.

(c) Is there overlap between the FIRST sets of any two right-hand sides of the same non-terminal?

(d) Is the grammar left-recursive (either directly or indirectly)?

(e) Is the grammar LL(1)? This is mainly based on the answers to item 3 and 4, but if there are $\epsilon$-productions, you need to also consider the FOLLOW sets as discussed in the second part of the LL(1) theorem on slide 22 of lecture 9. (Hint: this last issue comes up only for grammar $G_4$.)

1. (5 pts)

$G_1$: $S \rightarrow bab \mid bA$
$A \rightarrow d \mid cA$

2. (5 pts)

$G_2$: $S \rightarrow bC$
$C \rightarrow ab \mid A$
$A \rightarrow d \mid cA$

3. (5 pts)

$G_3$: $S \rightarrow AB$
$A \rightarrow Ca \mid \epsilon$
$B \rightarrow BaAC \mid c$
$C \rightarrow b \mid \epsilon$

4. (5 pts)

$G_4$: $S \rightarrow AB$
$A \rightarrow Ca \mid \epsilon$
$B \rightarrow cD$
$D \rightarrow aAcD \mid \epsilon$
$C \rightarrow b \mid \epsilon$

5. (10 pts) The following grammar represents the part of a grammar for statements in a programming language that includes if statements and function calls. (We've used $S$ for "statement," $F$ for "function call," $I$ for "if statement," $E$ for "expression," and $A$ for "argument list.") The tokens are `if`, `then`, `else`, `(`, `)`, `IDENT`, and `,`.

$S \rightarrow F \mid I$
$F \rightarrow$ `IDENT ( ` $A$ `)`
$I \rightarrow$ `if (` $E$ `)` $S \mid$ `if (` $E$ `)` $S$ `else` $S$
$E \rightarrow$ `IDENT`
$A \rightarrow$ `IDENT` $\mid$ `IDENT ,` $A \mid \epsilon$

This grammar is not LL(1), but can perhaps be made LL(1) by left-factoring. Here is what we get if we left-factor the $I$ rules:

$S \rightarrow F \mid I$
$F \rightarrow$ `IDENT (` $A$ `)`
$I \rightarrow$ `if (` $E$ `)` $S\ I'$
$I' \rightarrow$ `else` $S \mid \epsilon$
$E \rightarrow$ `IDENT`
$A \rightarrow$ `IDENT` $\mid$ `IDENT ,` $A \mid \epsilon$

Give the grammar after left-factoring the rules for $A$.

6. (5 pts) Now that the grammar is left-factored, it still isn't LL(1). In fact, it can't be LL(1) because it is ambiguous. (We discussed this ambiguity in homework 4. Left-factoring has no effect on ambiguity; both the original and left-factored grammars are ambiguous.) Here is an example of a sentence that can be parsed in multiple ways: `if (x) if (y) a() else b()`. Show that this is ambiguous by giving two parse trees (*using the second grammar from problem 5*) for that sentence.

   *Aside*: No ambiguous grammar can be LL(1), which means that any ambiguous grammar must violate one of the LL(1) conditions. It is easy to see that the above grammar is not left-recursive and, once left-factored, does not have any overlap between FIRST sets of right-hand sides of any non-terminals. So it must, and does, violate the FOLLOW set rule. Here is how it violates it: `else` is in FOLLOW($I'$), and is also in FIRST(`else` $S$), which is a right-hand side for $I'$ different from the one that is nullable. (Thus, when looking for an $I'$ and seeing `ELSE`, `parseI'` cannot tell which production to use.) It is a little hard to see how `else` is in FOLLOW($I'$); see if you can figure it out; we supply the answer at the end of the assignment.

7. (15 pts) The following grammar represents semicolon-separated lists of identifiers and integers:

$L \rightarrow E$ `;` $L \mid E$
$E \rightarrow$ `ID` $\mid$ `INT`

This one also needs left-factoring, which produces:

$L \rightarrow E\ L'$
$L' \rightarrow$ `;` $L \mid \epsilon$
$E \rightarrow$ `ID` $\mid$ `INT`

This grammar is, in fact, LL(1). It is easy to see that the FIRST sets of the right-hand sides for $L'$ are disjoint, and similarly for $E$. Write the functions for a top-down recognizer for this grammar. This is a function of type `token list` $\rightarrow$ `token list`, as in class. The tokens are given by this type:

```
type token = ID of string | INT of int | SEMIC | EOF
```

You should raise `SyntaxError` (again as in class) when the input is determined to be in error.

# 5 Answer to question in the "aside"

We show how `else` is in FOLLOW($I'$). The underlined `else` immediately follows the underlined $I'$ in the frontier of this parse tree: