COMS W4115
Programming Languages and Translators
Homework Assignment #2
Submit solutions electronically on
Courseworks/COMSW4115/Assignments
by 2:40pm, March 6, 2013

Instructions

- Problems 1-4 are each worth 25 points.
- You may discuss the questions with others in the class but your answers must be in your own words and your own code. You must not copy someone else's solutions. If you consult others or use external sources, please cite the people or sources in your answers.
- Solutions to these problems will be posted on Courseworks on March 11, 2013.
- This assignment may submitted electronically on Courseworks by 2:40pm, March 11, 2013 for 50% credit.
- Pdf files are preferred.

Problems

1. Interactive desk calculator for boolean nor-expressions.
   a. Construct a grammar that generates boolean nor-expressions containing the logical constants `true` and `false`, the left-associative binary boolean operator `nor` [where `p nor q` means not (p or q)], and parentheses.
   b. Show the parse tree according to your grammar for the nor-expression
      `true nor true nor (false nor false)`
   c. Implement an interpreter that takes as input newline-terminated lines of boolean nor-expressions and produces as output the truth value of each expression. You can use lex and yacc or their equivalents to implement your interpreter. Show the source code for your interpreter and the sequences of commands you used to test it.
   d. Run your interpreter on the following two inputs and show the outputs:
      (a) `(true nor false) nor (true nor false)`
      (b) `true nor true nor (false nor false)`

2. Infix to stack machine code translator
   a. Consider the Yacc specification in Fig. 4.59 of ALSU (p. 292). Modify this translator to produce stack machine code for each input line. For example, for the input "1*(2+3)" your translator should produce the instructions

```
push 1
push 2
push 3
add
multiply
done
```

   b. Implement your translator in Yacc (or its equivalent) and show the stack machine code generated for each of the following inputs:
      (i) `1+2*3`
      (ii) `1+(2-3)`
      (iii) `1+2-+3`
      (iv) `1+2--3`

3. Let *L* be the language generated by the following grammar:

```
S â†’ a S b S | b S a S | Îµ
```

   a. What language does this grammar generate?
   b. Show that this grammar is ambiguous.
   c. Construct the predictive parsing table for this grammar.
   d. Construct an LL(1) grammar for *L*.
   e. Construct the predictive parsing table for your grammar.

4. Let *L* be the language of pure lambda calculus expressions generated by the following grammar:

```
E â†’ ^ v . E  | E E | ( E ) | v
```

The symbols `^, ., (, )` and `v` are tokens. `^` represents lambda and `v` represents a variable.

An expression of the form `^v.E` is a function definition where `v` is the formal parameter of the function and `E` is its body.

If *f* and *g* are lambda expressions, then the lambda expression *fg* represents the application of the function *f* to the argument *g*.

a. Show that this grammar is ambiguous.

b. Construct an unambiguous grammar for *L* assuming that function application is left associative, e.g., *fgh* = (*fg*)*h*, and that function application binds tighter than `.`, e.g., (^*x*. ^*y*. *xy*) ^*z*.*z* = (^*x*. (^*y*. *xy*)) ^*z*.*z*.

c. Using your grammar, construct a parse tree for the expression (^*x*. ^*y*. *xy*) ^*z*.*z*.

d. Using the command `yacc -v` on a file containing your grammar, show the LALR(1) parsing action and goto table for your grammar.

---

aho@cs.columbia.edu