

CS 515 Programming Languages and Compilers I

Problem Set 1

This homework will not be graded. A sample solution will be made available at a later time.

Problem 1 – Ambiguous Grammars

Give an example of an ambiguous grammar. Justify your answer by giving a formal argument.

Problem 2 – Predictive Parsing

Assume the following CFG for predicate logic prefix expressions.

```
Start ::= Expr
Expr ::= or Expr Expr
Expr ::= and Expr Expr
Expr ::= xor Expr Expr
Expr ::= not Expr
Expr ::= Opnd
Opnd ::= true
Opnd ::= false
```

1. Give the FIRST sets for every right-hand side of the rules listed above. Give the FOLLOW sets for non-terminals *Start*, *Expr*, and *Opnd*.
2. Show that the grammar is LL(1). Show the transition table M.
3. Show the content of the stack and input of the table-driven LL(1) parser (see lecture 1, page 42) for each step during the top-down parsing of the input string
and not true or true false
4. Write a recursive descent parser in pseudo code (see pseudo code example: lecture 1, page 44), for the grammar above. Assume that the scanner provides the current token in global variable **token**. To request the next (or first) token, use the function **next_token**. Use a call to routine **error** to signal an error condition. No error recovery is required.
5. Show the content of the runtime stack during the parsing process for input string
and not true or true false

6. Implement your recursive descent parser in either C, C++, or Java. The code must run on the ilab machines. You may assume that all inputs to your parser are syntactically correct, i.e., you do not have to worry about errors here. The input program should be a single line (no new lines) stored in file `in.truth`.

This parser will be the basis for the implementation of an interpreter and a compiler for our language.

- **Interpreter** - Computes and prints the truth value obtained by evaluating the input program.
- **Compiler** - Generates code that when executed prints the truth value of the input program. The code should be written to the file `code.out`.

For example, for the input program

and not true or true false

Your interpreter should produce the output “RESULT: false”. Your compiler should produce RISC code in the file `code.out`. Every instruction that computes a new value, i.e., generates a value in a target register should use a “fresh” register. An sample code for our example input is as follows:

```
loadI true => r1
not r1 => r2
loadI true => r3
loadI false => r4
or r3, r4 => r5
and r2, r5 => r6
output r6
```