Language and Logic Assignment 2

- Please submit your solutions in a single .pdf file in addition to commented code, your lexer, grammar and executable files with instructions regarding execution.
- Submit each file separately in uncompressed format.
- The deadline for submissions is 18 May 2019.
- Only typed solutions will be accepted. Please show the steps when solving the questions.
- *Reports that do not comply with the aforementioned requirements will not be considered.*

Question 1.

20 points

For each of the following languages, design a CFG that recognizes it:

- $L = \{n \mid n \ mod \ 4 = 3\}$ where n is an unsigned binary number. Note that unsigned binary numbers are non-negative integer numbers. The alphabet is $\Sigma = \{0, 1\}$.
- $L = \{n \mid 15 < n\}$ where n is an unsigned binary number. Consider the alphabet $\Sigma = \{0, 1\}$.
- All strings where at least n a's occur before the n-th b, for example it should accept ab or $aacbcacbab$. The alphabet is $\Sigma = \{a, b, c\}$.
- $L = \{a^m b^n c^k \mid n, m, k \geq 0, (m = n) \text{ or } (n = k)\}$ with the alphabet $\Sigma = \{a, b, c\}$.

Question 2.

15 points

What languages do the following grammars generate? Show the steps, i.e. the possible derivation patterns.

- $\Sigma = \{0, 1, z, w\}$ and S is the start symbol
  S $\rightarrow$ 0Sw | X | Y
  X $\rightarrow$ 0Xz | O
  Y $\rightarrow$ 1Yw | O
  O $\rightarrow$ 1Oz | ϵ

- $\Sigma = \{0, 1\}$ and S is the start symbol
  S $\rightarrow$ A0A
  A $\rightarrow$ A A | 0A 1 | 1A0 | 0 | ϵ

Question 3.

20 points

- Is the following grammar LL(1)? Argue why.

  S $\rightarrow$ O | A | I | ID
  A $\rightarrow$ A $\wedge$ O | ID | (A) | N | I
  O $\rightarrow$ O $\vee$ A | ID | (O) | N | I
  N $\rightarrow \neg O \mid \neg A \mid \neg ID \mid \neg I$
  I $\rightarrow$ A $\implies$ O | O $\implies$ A | ID
  ID $\rightarrow$ a | b | c | $\top$ | $\bot$

- If it's not LL(1), transform it to an equivalent LL(1) grammar (i.e. it generates the same language as that of the above grammar).

Question 4.

15 points

Design a Turing Machine that reads a sequence of operations and performs them on the binary string that follows. This machine should also print the max number of 1s that appeared in the intermediate results. The possible operations include:

1. Bit Flipping (BF).
   Bit Flipping corresponds to inverting all bits in the string.
   i.e. 010101 becomes 101010

2. Left Shifting by 1 (LS). Left Shifting means moving the whole string one place to the left, truncating the first bit and adding a 0 to the end. i.e 010101 becomes 101010.

Notes:

1. The operations can appear in any order.
2. There can be an arbitrary number of operations in the beginning of the tape, they must be performed in sequence from left to right.
3. Feel free to use any numerical representation you find most suitable for counting the occurrences of 1's(i.e. binary, unary etc).

An example of the input is: BFLS,10101010 which requires that the string be bit-flipped then left shifted and finally print the maximum number of 1's occurred.

- **Explain informally how your Turing machine operates**
- **Design the Turing machine you just described**

Question 5.

30 points

Protocol buffers are a language, platform neutral mechanism to serialize structured data. A simplified specification of the protocol buffers language is as follows:

- A letter is any letter in the English alphabet small or capital.
- A number consists of decimal digits.
- An identifier is described by a letter followed by a sequence of letters, numbers or '_' i.e. a_b
- enum names, field names, service names and rpc(remote procedure call) names are identifiers.
- a message type or an enum type can start with any number of identifiers that are followed by a '.' and then a mandatory message name or enum name respectively.

- The language contains statements, and all statements are terminated by ';'.
- A package declaration is a statement that starts with "package" followed by an identifier and ending with ";"
- Types are one of "num", "bool", "string", "bytes", a message type or an enum type.
- A bool is either "true" or "false".
- A string consists of any number of leters inside quotes "".
- A field starts with a type is followed by a field name and '=' and ends with a number,a bool or a string.
- An enumeration starts with "enum" and is followed by an enum name and an enum body.
- An enum body starts with '{' ends with'}' and contains any number of enum field statements within.
- an enum field statement consists of an identifier, '=' and a number.
- A message starts with "message" and is followed by a message name and a message body.
- A message body starts and ends with '{' and '}' and can contain any number of fields, enums, types, or even nested messages.
- A service starts with "service" is followed by the service name which is an identifier and a service body.
- A service body starts and ends with '{' and '}' and can contain rpcs.
- An rpc statement starts with "rpc" is followed by a rpc name, a message type in parentheses "()" and "returns" another message type in parentheses again ending with a ";"
- Every protocol buffers file starts with a version statement starting with "version", "=", followed by a string and ending with ";" and is followed by any number of enums messages or services in any order and optionally a single package declaration.

An example of how the language looks like is as follows:

```
version = "proto";
enum ProcessState {
  UNKNOWN = 0;
  STARTED = 1;
  RUNNING = 2;
}
message outer {
  enum IsValid{
  NO = 0 ;
  YES = 1 ;
  }
  message inner {
    num myval = 1;
  }
  string s = "amessage";
  num magic = 1337;

}
service OuterInner {
  rpc extractInner (outer) returns (outer.inner);
}
```

- **Design a Context Free Grammar for the above Language**
- **Remove any ambiguity, if there is any in your designed grammar.**
- **Implement a recursive descent parser for the language in a programming language of your choice**
- **Implement a parser by using ANTLR or any other parser generator i.e. Bison**