# Homework 6

**6CCS3CFL - Compilers & Formal Languages**

Finley Warman

November 2, 2020

## Contents

**Question 1** **2**

**Question 2** **2**

**Question 3** **3**

**Question 4** **3**

**Question 5** **3**

**Question 6** **4**

---

## Question 1

Q:

- (i) Give the regular expressions for lexing a language consisting of whitespaces, identifiers, numbers, operations `=, <, >`, and the keywords `if, then, else`.
- (ii) Decide whether the following strings can be lexed in this language?

A:

```
DIGIT       = RANGE("0123456789")
START_DIGIT = RANGE("123456789")
NUMBER      = OPT(CHAR('-')) . (DIGIT + (START_DIGIT . DIGIT*))

WHITESPACE  = RANGE(" \n\r\t")

OPERATOR    = CHAR('=') + CHAR('<') + CHAR('>')

KWORDS      = "if" + "then" + "else"

LOWERCASE   = RANGE("abcdefghijklmnopqrstuvwxyz")
ID          = LOWERCASE . LOWERCASE* . DIGIT*

LANG        = ("nm":NUMBER) + ("op":OPERATOR) + ("kw": KWORDS)
                 + ("id":ID) + ("sp":WHITESPACE)
```

- a) `if y4 = 3 then 1 else 3` - YES

  ```
  kw:if,sp: ,id:y4,sp: ,op:=,sp: ,kw:then,sp: ,nm:1,sp: ,kw:else,sp: ,nm:3
  ```

- b) `if33 ifif then then23 else else 32` - YES

  ```
  id:if33,sp: ,id:ifif,sp: ,kw:then,sp: ,id:then23,
  sp: ,kw:else,sp: ,kw:else,sp: ,nm:32
  ```

- c) `if x4x < 33 then 1 else 3` - YES

  ```
  kw:if,sp: ,id:x4,id:x,sp: ,op:<,sp: ,nm:33,sp: ,
  kw:then,sp: ,nm:1,sp: ,kw:else,sp: ,nm:3
  ```

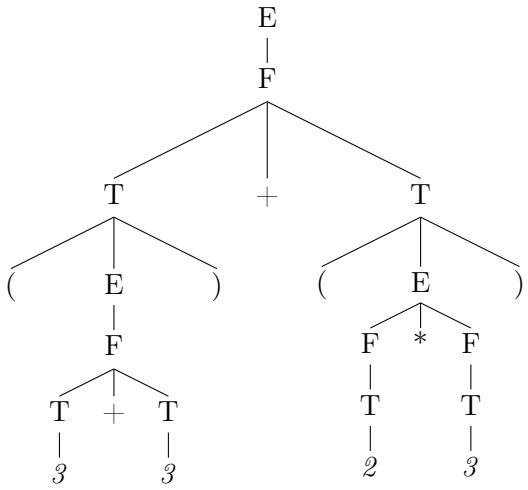
## Question 2

Q: Suppose the grammar:

```
E -> F | F.*.F | F.\.F
F -> T | T.+.T | T.-.T
T -> num | (.E.)
```

Where E, F and T are non-terminals, E is the starting symbol of the grammar, and *num* stands for a number token.
Give a parse tree for the string `(3+3)+(2*3)`

A:




## Question 3

Q: What is an ambiguous grammar? Give an example.

A: An ambiguous context-free grammar is one for which there is some string that has more than one valid parse-tree, i.e. there is more than one way to derive the string in terms of the grammar.
An example of an ambiguous grammar:

```
E := N
E := E.+.E
N := 1 | 2

e.g. 1 + 2 + 11
Can be parsed as (1+2)+11 or 1+(2+11)
```

## Question 4

Q:

- i) Give a grammar that can recognise all such boolean expressions,
- ii) Give a sample string involving all rules, that can be parsed by this grammar.

A:

```
B -> P | A | N | O
A -> true | false
O -> B.∧.B | B.∨.B
P -> (.B.)
N -> ¬.B

(B is starting symbol of grammar)
```

ii) $\neg(true \wedge false) \vee true$

## Question 5

Q: What is the purpose of atomic parsers, and semantic actions?

A:

- Atomic parsers take an input, and do a simple transformation. These effectively act as a base-case for our parser combinator, allowing us to define alternative, sequence, and semantic action parsers in terms of them.
  This makes it easy to write grammars in terms of code, as we can treat each nonterminal symbol as its own atomic parser.

- Semantics Actions offer a way to collapse parse trees into a simplified form, that isn't determined entirely by the shape of the grammar (Effectively, 'cutting out' needless nodes).
  They are functions acting as simplification rules over the grammar, applied to the result of a parser.
  e.g. converting strings-numbers into integer-numbers (to make them 'useful'), or by combining tokens representing the sum of two numeric literals `N.op.N -> 1+2` into the evaluated `op(N,N) -> +(1,2) = 3` by applying `op -> +` for expressions of this form.

## Question 6

Q: Advantages of first lexing a string, then feeding the input as a sequence of tokens into a parser?

- We can recognise invalid tokens and throw an error (cheap: regular expressions) before wasting effort on parsing (expensive: context-free grammar).

- It is easier to design a grammar in terms of tokens and process a token string, rather than design this token recognition into the grammar.

- As a result of the above, it is easier to avoid ambiguity in your grammar when you are only dealing with valid tokens.

- Keeps things simple! This makes developing a language easier to debug, and easier to make changes.