Name : Aditi Singla
Entry No. : 2014CS50277

COL703: Logic for CS

Assignment 1 : Scanning and Parsing P0

Signature for AST structure

```
signature AST = sig
        datatype Prop = TOP
                      | BOTTOM
                      | ATOM of string
                      | NOT of Prop
                      | AND of Prop * Prop
                      | OR of Prop * Prop
                      | IFF of Prop * Prop
                      | IMP of Prop * Prop
                      | ITE of Prop * Prop * Prop
        val toPrefix    : Prop -> string
        val toPostfix   : Prop -> string
        val isEqual     : Prop -> Prop -> bool
end
```

Symbols

Terminals : TOP, BOTTOM, ATOM, NOT, AND, OR, IFF, IF, THEN, ELSE, SPACE, RIGHTP, LEFTP

Non Terminals : stmt, stmtA, stmtB, stmtC, stmtD, stmtE, stmtF, alist (of tokens)

Start : Start

*Note : RIGHTP and LEFTP correspond to the right and left parenthesis, respectively.

Precedence Order

Based on the discussion about ‘if then’ and ‘if then else’, the precedence order has been modified and is as follows:

> **RIGHTP, LEFTP > NOT > AND > OR > IF THEN ELSE > IF THEN > IFF**

Grammer

Start -> stmt | SPACE stmt

stmt -> stmtA IFF stmt | stmtA

stmtA -> IF stmtA THEN stmtA | IF stmtA THEN stmtB ELSE stmtA | stmtC

stmtB -> IF stmtA THEN stmtB ELSE stmtB | stmtC

stmtC -> stmtC OR stmtD | stmtD

stmtD -> stmtD AND stmtE | stmtE

stmtE -> NOT stmtE | stmtF

stmtF -> TRUE | FALSE | alist | LEFTP stmt RIGHTP

alist -> ATOM SPACE alist | ATOM alist | ATOM SPACE | ATOM

Explanation

- For managing the precedence order of ‘if then’ and ‘if then else’ in the grammar, a careful case analysis has been done. ‘stmtA’ covers all the cases with or without ‘else’ and also cases with no ‘if then/if then else’ at all. While ‘stmtB’ is meant for cases only where each ‘if then’ must come with ‘else’ or there must be no ‘if then/if then else’ at all. These two cases help us define the derivation for both of them.
- In the specifications provided to the lexer, for each of the strings which map to one of the type in the Datatype defined, a token is produced.
- For every whitespace of length >2, a token is created with the length as its value. Since the whitespaces following any of the token matching one in the Datatype are taken care of by the token itself, these cover the ones between strings (of type ATOM) which are then later used to maintain **space sensitivity of identifiers** in this grammar.
- All the strings, which will contribute to identifiers (ATOM), are being read separately by the lexer (the spaces in between are stored into another token), the grammer is such that the last rule for ‘alist’ attempts to join all such strings to form one single string for the ATOM.
- Each line in the input file is converted to it’s corresponding AST and then appended to a list. This list is later read, to print the prefix and postfix for each AST in a file. Note that any

line which gives an error in parsing is ignored and no output is printed in the file. Though we can observe the error message for each such line on console.

References

- Example of calc, obtained as an example in the ML-YACC library folder.
    - https://packages.ubuntu.com/trusty/amd64/ml-yacc/filelist
- http://cs.wellesley.edu/~cs235/fall08/lectures/35_YACC_revised.pdf
- http://www.cs.tufts.edu/comp/181/ug.pdf