COL703: Logic for Computer Science	September 8, 2017

Assignment 1

Problem: Scanning and Parsing $\mathcal{P}_0$

Implement a scanner-parser in SML which takes in a sentence in $\mathcal{P}_0$ and converts it to an Abstract Syntax Tree (AST). The signature for the AST structure (which you also have to implement) is as follows.

```
signature AST = sig
    datatype Prop = TOP                           | BOTTOM
                  | ATOM of string                | NOT of Prop
                  | AND of Prop * Prop            |  OR of Prop * Prop
                  | IMP of Prop * Prop            | IFF of Prop * Prop
                  | ITE of Prop * Prop * Prop

    val toPrefix    : Prop -> string
    val toPostfix   : Prop -> string
    val isEqual     : Prop -> Prop -> bool
end
```

`IMP` corresponds to a conditional, `IFF` to a biconditional and `ITE` to an if-then-else operator. Prefix and postfix representations will be space separated strings of atomic and constructor identifiers in the appropriate order.

The formulation of $\mathcal{P}_0$ used will **NOT** be fully parenthesized. In this case, you will have to use association rules and precedence order to unambiguously determine the AST. The table below lists the symbols in $\mathcal{P}_0$ and their text representations, i.e., how they will appear in the input. Operators are listed in descending precedence order.

Operators in $\mathcal{P}_0$			
Name	Symbolic example	Text Representation	Associativity
Negation	$\neg s$	`NOT s`	–
Conjunction	$a \wedge b$	`a AND b`	Left
Disjunction	$a \vee b$	`a OR b`	Left
Conditional	$p \rightarrow q$	`IF p THEN q`	Right
Biconditional	$p \leftrightarrow q$	`p IFF q`	Right
if-then-else	$p \ ? \ q : r$	`IF p THEN q ELSE r`	–
Other symbols in $\mathcal{P}_0$			
Top	$\top$	`TRUE`	–
Bottom	$\bot$	`FALSE`	–
Parenthesis	(s)	`(s)`	–

For further clarification, students can refer to Section 2.3 (Associativity and Precedence) of the slides.

The symbols `a,b,p,q,r,s` are placeholders for text representations of sentences a, b, p, q, r, s respectively in $\mathcal{P}_0$. Atoms in the input can be anything (string, integer etc.) except whitespace and keywords. The input text is whitespace-insensitive, except for the identifier of an atom.

You are free to write your own code from scratch (given the relatively small size of the grammar). Otherwise, you can make use of these lexers - ML-ulex, ML-Lex and these parsers - ML-Antlr, ML-Yacc. Input examples:

- `(do OR NOT do) AND NOT there is try.`
- `IF you strike me down THEN I shall become more powerful than you can possibly imagine.`
- `IF 589 THEN IF 241 THEN Cheshire Cat ELSE Mad Hatter IFF March Hare OR Caterpillar`
- `IF 34 THEN 256 ELSE 999 OR NOT 65536`

The dangling-else problem can be resolved by binding the `ELSE` clause to the closest unmatched `IF` condition that precedes it lexically. That way it is both LL(1) "parseable" as well as LALR(1) "parseable". Then it doesn't matter whether you do top-down or bottom-up parsing. Some rough explanatory examples:

- `IF a THEN IF b THEN c ELSE d`
  is parsed to
  `IFTHEN(a,IFTHENELSE(b,c,d))`

- `IF a THEN IF b THEN IF c THEN d ELSE e`
  is parsed to
  `IFTHEN(a,IFTHEN(b, IFTHENELSE(c,d,e))`

- `IF a THEN IF b THEN IF c THEN d ELSE e ELSE f`
  is parsed to
  `IFTHEN(a,IFTHENELSE(b, IFTHENELSE(c,d,e), f)`

Input/Output Specifications

Input Text file with a test case in each line.

Output Text file with two lines corresponding to each test case in the input file - a prefix representation in the first line and a postfix representation in the second.

The command used will be `sml <filename> <inputfile> <outputfile>` .

You are also required to submit a PDF report detailing exactly the grammar that you developed to parse the input.