**First Order Logic Parser**

**Introduction**

The aim of this project is to implement a parser for First Order (FO) Logic. Based on the given sets of variables, constants, predicates, equality, connectives and quantifiers, it produces the corresponding grammar for the language of valid FO formulae. For a valid input formula, it generates the corresponding parse tree

**Prerequisites**

The program uses Python 3.7.4.

The package nltk is also required. In order to check if this is install on the current machine, run the *pip3 list* command in the terminal. If nltk is not included in this list, run the *sudo pip3 install -U nltk* command.

**Description**

The program requires an input text file (.txt), which contains the sets of variables, constants, predicates, equality, connectives, quantifiers and a (possibly non-valid) formula. Upon reading this file, the program outputs on the screen the corresponding grammar for the language of valid formula. For a valid formula, it generates a parse tree, and outputs it to a ParseTree.ps file. The program also maintains a server.log file, which indicates whether the input file is of the correct format and whether the formula is valid/invalid.

The log file displays information with regards the validity of the file. A valid file should contain have the correct number of elements for the sets of equality, connectives and quantifiers, i.e. 1, 5, and 2, respectively. If the input file doesn’t fulfil this requirement, the corresponding error message displayed in the log file would be one of *“Invalid number of equality symbols”*, “*Invalid number of connectives”*, or “*Invalid number of quantifiers*”. Moreover, a valid file should have different symbols for constants, predicates and variables. If the input file fails to do so, the log file would display one of the error messages: *“Invalid constant name”*, *“Invalid predicate name”*, or *“Invalid variable name”*.

If the input file is valid, the log file will display the *“Valid file”* message. The log also contains information about the formula. In case of an invalid formula, the log will display *“Invalid formula: ”*, followed by one of the error messages: *“unmatching brackets found*”, *“invalid equality found”*, *“invalid predicate found”*, *“invalid symbol found after quantifier”*, or *“redundant brackets found”*.

Some examples of the inputs/outputs for the program can be seen below.

An input file with the content below would be considered valid. The given formula is also valid:

*variables: w x y z*

*constants: C D*

*predicates: P[2] Q[1]*

*equality: =*

*connectives: \land \lor \implies \iff \neg*

*quantifiers: \exists \forall*

*formula: \forall x ( \exists y ( P(x,y) \implies \neg Q(x) )*

*\lor \exists z ( ( (C = z) \land Q(z) ) \land P(x,z) ) )*

The grammar generated by the program is:

*Terminal symbols: \land \lor \implies \iff \neg \exists \forall = C D w x y z P Q ( ) ,*

*Non-terminal symbols: Start Formula PredicateFormula EqualityFormula T connective negation quantifier equality constant variable predicate o\_bracket c\_bracket separator*

*Production rules:*

*Start -> Formula*

*Formula -> PredicateFormula | EqualityFormula | o\_bracket Formula connective Formula c\_bracket | negation Formula | quantifier variable Formula*

*PredicateFormula -> predicate o\_bracket variable separator variable separator ... separator variable c\_bracket*

*EqualityFormula -> bracket variable equality variable bracket | bracket constant equality constant bracket | bracket constant equality variable bracket | bracket variable equality constant bracket*

*T -> constant | variable*

*connective -> \land | \lor | \implies | \iff*

*negation -> \neg*

*quantifier -> \exists | \forall*

*equality -> =*

*constant -> C | D*

*variable -> w | x | y | z*

*predicate -> P | Q*

*o\_bracket -> (*

*c\_bracket -> )*

*separator -> ,*

The corresponding parse tree can be visualised in the figure below.



Another example of a valid file with a valid formula is:

*formula: A price E cost1 ( Same(cost1, price) AND ( NOT Non\_zero(price) IFF (cost1 == 30) ) )*

*equality: ==*

*connectives: AND OR IMPLIES IFF NOT*

*quantifiers: E A*

*variables: price cost1*

*constants: 30 Z*

*predicates: Same[2] Non\_zero[1] notEqual[3]*

The generated grammar is:

*Terminal symbols: AND OR IMPLIES IFF NOT E A == 30 Z price cost1 Same Non\_zero notEqual ( ) ,*

*Non-terminal symbols: Start Formula PredicateFormula EqualityFormula T connective negation quantifier equality constant variable predicate o\_bracket c\_bracket separator*

*Production rules:*

*Start -> Formula*

*Formula -> PredicateFormula | EqualityFormula | o\_bracket Formula connective Formula c\_bracket | negation Formula | quantifier variable Formula*

*PredicateFormula -> predicate o\_bracket variable separator variable separator ... separator variable c\_bracket*

*EqualityFormula -> bracket variable equality variable bracket | bracket constant equality constant bracket | bracket constant equality variable bracket | bracket variable equality constant bracket*

*T -> constant | variable*

*connective -> AND | OR | IMPLIES | IFF*

*negation -> NOT*

*quantifier -> E | A*

*equality -> ==*

*constant -> 30 | Z*

*variable -> price | cost1*

*predicate -> Same | Non\_zero | notEqual*

*o\_bracket -> (*

*c\_bracket -> )*

*separator -> ,*

The generated parse tree is shown below.

****

**Objectives / how to run**

The program is compiled by running the command line instruction python3 input.txt (if input file is not specified example1.txt is set as default input), in the working directory.