Compiler Design Summative Assignment: Source Code Documentation

1. Program Instructions

a. Install relevant Python modules

Using pip other installer program (e.g. Homebrew), install the modules necessary to run the program.

This can be accomplished with the following command: python -m pip install SomePackage

```
from datetime import datetime
import time
import sys
import string
import random

#Parse tree visualization libraries
import anytree
import graphviz
from anytree import Node, RenderTree
from anytree.dotexport import RenderTreeGraph
from anytree.exporter import DotExporter
```

Of particular note are the modules utilized for parse tree construction and visualization, which are not part of the standard Python library: *anytree* and *graphviz*.

Installing anytree: pip install anytree

Installing graphviz:

On Windows:
i. Download the stable Graphviz version:
https://graphviz.gitlab.io/_pages/Download/Download_windows.html
ii. Set the PATH variable: path="...;c:\Program Files (x86)\Graphviz2.38\bin;"
iii. Install Graphviz library using: pip install graphviz

b. Running the program

Upon navigating to the appropriate directory containing the Python program (compilerdesign.py) and intended input file (e.g. example.txt), start the program with the command: python compilerdesign.py example.txt

The name of the input file should be supplied as the first command line argument.

As the program is running, any encountered errors are printed to the console and logged in a log file, logfile.log. If a log file does not yet exist in the current directory, it is created the first time the program is run; otherwise, entries are appended to the log with each subsequent execution of the program. Each tab-delimited entry in the log file contains a time stamp, name of the relevant input file, status (ERR/OK), and appropriate error message.

In the event that the program deems a supplied input file and corresponding grammar/FO formula invalid, the encountered errors are logged and printed to the console. Otherwise, if the grammar and FO formula are valid, 2 files are output:

timestamp + *inputfilename* + *outputgrammar.txt* : a text file containing the corresponding grammar for the supplied language

timestamp + *inputfilename* + *outputparsetree.png* : an image containing the parse tree visualization of the supplied input formula

c. Grammar, Non-Terminals Key

In the parse tree visualization, non-terminals are **represented with an asterisk** (e.g. F*) to distinguish them from potential terminals represented by the same token (e.g. a constant 'F' and non-terminal 'F*'). This enables the parser to successfully parse formulas in which terminals happen to take the same letter as a non-terminal in my grammar, without needing to designate all capital letters representing non-terminals as forbidden tokens.

Forbidden tokens otherwise include '$', which is used to represent the EOF symbol when parsing. The standalone symbols ( ) , are reserved strictly as terminals in the grammar.

Non-Terminals Key:

- F* : formula
- T* : term
- C* : connective
- Q* : quantifier
- P* : predicate expression
- Z* : predicate symbol
- J* : list of variables contained within predicate expression, separated by commas
- K* : constant
- V* : variable

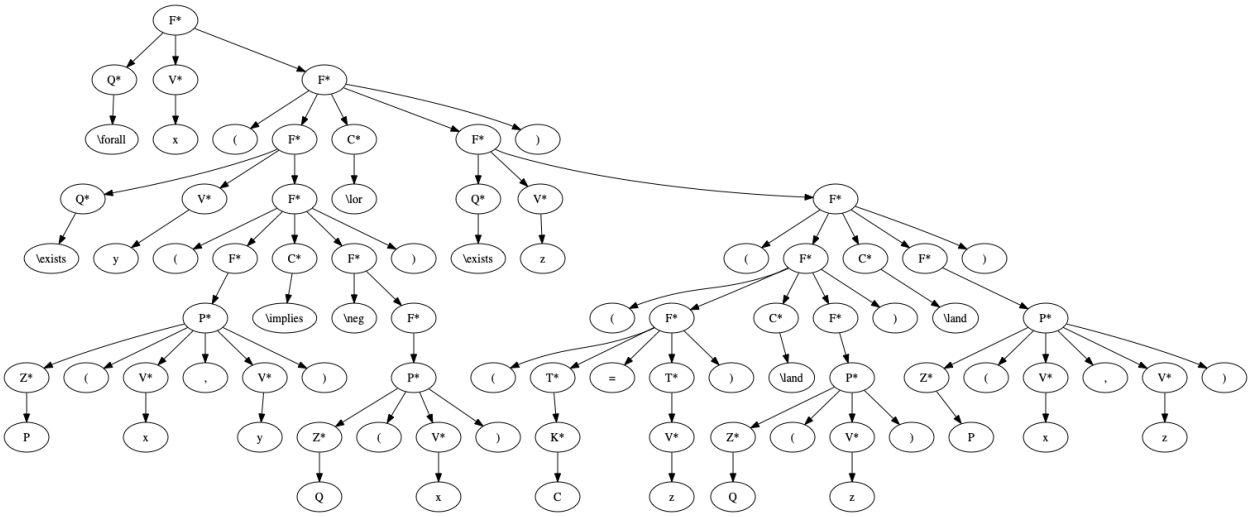
2. Examples

a. *example.txt*

Output grammar:

```
A formal grammar is defined as a quadruple (V_t, V_n, P, S), where:
- V_t is a set of terminal symbols
- V_n is a set of non-terminal symbols
- P is set of production rules
- S is the start symbol, which is a non-terminal

See program documentation for meaning of each non-terminal symbol in the grammar below.
The formal grammar for the supplied input file, example.txt, is defined by the sets V_t,
V_n, P, and S as follows:

V_t = {w, x, y, z, C, D, P, Q, \land, \lor, \implies, \iff, \neg, \exists, \forall, =}

V_n = {F*, P*, Z*, T*, V*, C*, K*, Q*, J*}

P = {

F* -> ( F* C* F* ) | Q* V* F* | \neg F* | ( T* = T* ) | P*
T* -> V* | K*
C* -> \land | \lor | \implies | \iff | \neg
Q* -> \exists | \forall
P* -> Z* ( J* )
J* -> V* | V* , J*
Z* -> P | Q
K* -> C | D
V* -> w | x | y | z

}

S = {F*}
```

Output parse tree visualization:



b. *example1.txt*

Output grammar: (beginning of file removed for brevity)

```
See program documentation for meaning of each non-terminal symbol in the grammar below.
The formal grammar for the supplied input file, example1.txt, is defined by the sets V_t,
V_n, P, and S as follows:

V_t = {price, cost1, 30, Z, Same, Non_zero, notEqual, AND, OR, IMPLIES, IFF, NOT, E, A,
==}

V_n = {F*, P*, Z*, T*, V*, C*, K*, Q*, J*}

P = {

F* -> ( F* C* F* ) | Q* V* F* | NOT F* | ( T* == T* ) | P*
T* -> V* | K*
C* -> AND | OR | IMPLIES | IFF | NOT
Q* -> E | A
P* -> Z* ( J* )
J* -> V* | V* , J*
Z* -> Same | Non_zero | notEqual
K* -> 30 | Z
V* -> price | cost1

}

S = {F*}
```

Output parse tree visualization: