FO Compiler Documentation

Setup

The source code is in the file **focompiler.py**.

To produce parse tree output as a PNG, you MUST have graphviz installed on your machine. You can find a download link for Windows here: https://graphviz.gitlab.io/_pages/Download/Download_windows.html.

This installer does not set the PATH to graphviz which the Python module requires so you must add it to the PATH. For me this was done by adding **C:\Program Files (x86)\Graphviz2.38\bin** to the PATH. Generally, it will be **<install directory>\Graphviz<version>\bin>**.

You must also install the python graphviz module. You may do this using **pip install graphviz**.

You must install the python anytree module. You may do this using **pip install anytree**.

The program also uses **sys, re, random, datetime** modules which are built in to Python so no further action should be required for these.

Running the compiler

Running the python file directly, e.g. in the Python IDLE will start the program and prompt you to enter the name of the file you wish to compile. This will be relative to the working directory so having the target file in the same directory is recommended for ease of use.

Alternatively, you can run the compiler from the command line and pass the target file as the first argument. An example command is **python focompiler.py test.txt** where test.txt is the file you wish to parse.

Input Format

The input file is expected to be in the same format as that given in the example that comes with the specification. See **test.txt** for an example of this.

Output Format

All output is logged to the file **focompiler_log.txt** in the directory the program executes in. This file is appended to for each file, though the file name that generated the log message is included in the log message.

The program will generate a grammar based on the input file. This will be output to a file named **grammar_<original file name>**. If the input file is **test.txt**, the grammar will be in **grammar_test.txt**.

The program will generate a PNG of the parse tree based on the input file. This will be output to a file named **parsetree_<original file name with extension stripped>.png**. If the input file is **test.txt**, the parse tree will be in **parsetree_test.png**. The stripped extension name is generated by splitting at occurrences of a period (**.**) and taking only the part of the file name before the first period. This means if a file has multiple extensions, all extensions will be stripped.

The program will also generate some basic information about successes/failures on stdout which will display on the console.

Forbidden Symbols

To avoid the need to restrict the use of strings that are identical to our non-terminals, all non-terminals are prefixed with colons (:) which are the delimiter used to parse the key/value pairs for declarations from the input file anyway. Using a colon anywhere but to delimit a key/value pair in the input will cause varying behaviour but will always be an error. The use of colons also makes non-terminals easier to see in the output of the parse tree.

Your input should not contain the string \0 (Python string '\\0') since this is used to denote the end of a formula and is automatically added to the end of any input formula. Using a \0 in the formula, as a quantifier for example, will end parsing early and an error will be output.

Types of error

The program will log errors to the log file when:

- Reading the input file fails
- Multiple colons exist on one declaration line in the input file
- The key specified on a declaration line in the input file is invalid or duplicated
- Any of the keys named **variables, constants, predicates, equality, connectives, quantifiers, formula** are completely missing from the input file
- Declared variables, constants, predicates, equality, connectives or quantifiers contain invalid characters for each of their respective types
- Predicate declarations are missing [or], the arity is missing, or the arity is not an integer
- Cardinality of equality is not 1
- Cardinality of connectives is not 5
- Cardinality of quantifiers is not 2
- Any of **variables, constants, predicates, equality, connectives, quantifiers** contain a non-unique symbol. This avoids ambiguity in formulae.

Syntax-specific errors:

- Unexpected token: when a terminal has been found but it doesn't match anything the current production expects. This will usually occur if your brackets are unbalanced or your formula terminates earlier than expected (which is the same thing technically). A position number will be provided. This is a position based on what the compiler considers to be tokens. Every whitespace in the formula is a separator, as are '(', ')' and the comma. This means that x(x is actually 3 tokens, despite being input without whitespace.
- Unrecognized token: when a formula contains a token that hasn't been declared. A token position is also output
- Syntax error: when a token has no valid production e.g. \forall \forall \forall where \forall is a quantifier doesn't exist in the grammar so would produce an error on the second \forall since no formula can be created matching that pattern. A token position is also output.
- Invalid number of predicate variables: when a predicate is used with more/less variables than its declared arity. A token position is also output.