First Order Logic Parser - Documentation

Dependencies and Installation

First and foremost, the parser requires Python version 3.6+ to run. It has been developed and tested on 3.7.x and 3.8.x

The parser relies on three imported packages from the Python standard library, `re`, `pathlib` and `sys`, and one external python package, `graphviz`.

The former three do not need special installation.

To use `graphviz`, please follow the instructions at the beginning of the following webpage: https://graphviz.readthedocs.io/en/stable/manual.html

In case of not being able to access this page, the instructions are paraphrased below: - Please first install Graphviz on your operating system, by choosing a download from the following webpage: https://www.graphviz.org/download/

- Once Graphviz is installed, please make sure your `/graphviz/bin/` directory is added to your system's `PATH` variable, so that the `graphviz` Python package can successfully use Graphviz's functions from the command line under the hood. You can check if this was successful by running `dot -V` from your command line outside of the folder in question - if done correctly, this command will print the version of your Graphviz installation.
- To finalise installation, run `pip install graphviz` from your command line.

Usage

First make sure your input file `filename.ext` is in the same working directory as the program `parser.py`.

To use the program, run the command `python ./parser.py ./filename.ext` from the directory containing the parser and the input file.

If the program `parser.py` is not called via the command line with the filename as an argument following it, it will not know what input to parse, and will exit.

Input

The parser expects as input a text file of at least 7 lines, with exactly 7 of these lines beginning with `setname:` to define some features of the grammar. Guidance supplied within the coursework specification and in the DUO FAQ on how to format the sets, which sets are allowed to be empty and what characters are allowed in members of each set has been followed - for example, the input will be invalid if members of a set are separated by anything but a single space, or if there are fewer than 5 entries on the "connectives:" line, of if the formula is split over multiple lines that are not consecutive.

General Program Flow

The parser takes an input file and ingests each line, making sense of the members of each set in the grammar and storing them accordingly. It then prints the grammar to a file (or, if for some reason it cannot print to a file, displays the grammar to the console) before starting to parse the token stream. It uses a simple recursive predictive parsing technique as described in lectures, taking advantage of the LL(k) nature of the First Order production rules. While parsing, the program builds the parse tree symbol by symbol. It finishes by rendering the tree and outputting a success message to the log file.

Outputs

If ran on a valid file, the parser will produce the following outputs:

- `grammar.txt`, a file describing the grammar of the language supplied in the input file
- `log.log`, a text file containing a message informing the user of successful parsing of the input file
- `ParseTree`, a side effect file of Graphviz's graph rendering that can be ignored
- `ParseTree.pdf`, a pdf containing the visual representation of the parse tree for the formula

In case of an error, `log.log` will always contain the error message that provides more detail on why the parser halted.

Depending on where the error occurred, the program may not have output the `ParseTree` and `ParseTree.pdf` files, say if the error is in the formula but the rest of the file was formatted correctly, or it may have also not output `grammar.txt`, say if there was a problem with the file formatting or the entries in a set of the grammar.

The parse tree is of a standard construction and aesthetic. Internal nodes, denoted by single bold letters, are non-terminal symbols, while leaves are terminal symbols.

Examples

Contained within the `examples` folder in the submission are a few examples of the program's outputs based on some devised inputs.

- `ex1` contains the outputs from a typical valid file, similar to the one provided with the coursework
- `ex2` contains the outputs from a file that has a valid grammar, but an error in its formula
- `ex3` contains the outputs from a file that has a valid formula, but an error in one of the predicate definitions