Propositional Logic Engine - User Manual

PLE v1.0.

Nicholas Killeen

Contents

1. Installing and Running the Program

1.1. Licence Agreement

This software is distributed under the *Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International Public License*, a copy of which is available at https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode.

1.2. Using the Portable Version

Supported Windows machines are able to run a standalone version of the solution, meaning that there is no necessary installation of Python or Python modules, negating issues of version control. It is important to recognise that the solution still runs through Python and its dependencies, but these are pre-installed to the distribution device.

In such a case that the Portable Version be used (which comes recommended; the epithet "Portable" is not meant to confer the relative superiority of the Non-Portable Version, and does not denote any functionality difference between the two types of distributions), the GUI can be simply ran through the shortcut "P.L.E" in the "PLE (Portable)/" directory, and the Command Line Version can be accessed likewise through the shortcut "P.L.E (Command Line)", in the same directory.

Success in executing the Portable Version sees that you skip to *Part 2. Using the Command Line Interface*.

1.3. Set up the Command Line Interface

In the event that the Portable Version is rendered unusable, particularly, if an Operating System outside of the supported Windows range is being used, both the CLI and GUI require the installation of Python. If it is known that the Graphical User Interface is not required to be installed for your purposes, Python version 3.4 or greater must be present for the CLI to run (previous versions may still work, but go unsupported). If it is expected that the Graphical User Interface be required, then the Python version must also be supported by the PySide module https://pypi.python.org/pypi/PySide/, while still being at least version 3.4, ;

Irrespectively, the CLI requires Python 3.4 or greater, as can be downloaded from https://www.python.org/downloads/. Once Python has been installed, the CLI can be started by moving to the directory "PLE (Source Only)/", and running the shortcut "P.L.E (Command Line)".

See *Section 2. Using the Command Line Interface* for how to operate the CLI.

1.4. Set up the Graphical User Interface

To create an environment in which the GUI can run, the requirements of *Section 1.3. Set up the Command Line Interface* must first be followed. Once the CLI is successfully running, the PySide package must be installed. This can be done by following the instructions on https://pypi.python.org/pypi/PySide/ (due to the perceived volatility of the PySide project, it may be best that version 1.2.4 only is installed if newer releases are available, but not asserted to contain backwards compatibility).

After PySide has been installed, executing the shortcut "P.L.E" within the directory "PLE (Source Only)/" will cause the GUI to be opened.

See *Section 2. Using the Graphical User Interface* for instructions specific to the use of the GUI.

2. Using the Command Line Interface

2.1. Commands and Arguments

On running the CLI, the user should be greeted with the screen:

```
Running PLE, welcome!
Press the return key twice to quit.

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>>
```

Snippet 2.1. A. Welcome Screen

To which it is expected that the user types 1, 2, 3, or 4. Pressing return to enter an empty input will cause the program to terminate, and the window to close. (Pressing the return key twice in succession at any point in the program will cause it to exit, but from this screen only one key press is necessary).

Accidentally providing an invalid input will result in the P.L.E asking the same question again, as follows:

```
Running PLE, welcome!
Press the return key twice to quit.

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution
>>> Hey! Crusader!
Did not recognise command.

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution
>>>
```

Snippet 2.1. B. Main Program Prompt after Receiving Invalid Input

In fact, typing invalid input to any prompt ">>>" will cause flow of control to return to this point, where the four possible Commands are listed and one of the four are expected. Such behaviour is quickly discernible by the empty new-line, just before the list is printed.

Each of the four Commands performs a different task. Command 1 allows facts to be appended to a database, which has a lifespan of the current program instance. Command 2 allows the deletion of added facts. Command 3 shows a list of all facts that are currently stored within the database. Command 4 allows the user to specify what information is to be requested. More detailed descriptions of each follow.

Command 1: Add a Fact

On choosing to add a fact, the program behaves as follows:

```
1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 1
Enter the fact to add:
>>>
```

Snippet 2.1. C. Prompt for Command 1: Add a Fact

The user must then provide a fact, in the form detailed in *Part 4. Writing and Requesting Facts*. The entry of an invalid fact will result in behaviour demonstrated in *Snippet 2.1. D.*, and a valid fact will result in what can be observed in *Snippet 2.1. E.*.

```
Enter the fact to add:
>>> Have you any nuts?
Failed to append fact: fact is malformed.

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>>
```

Snippet 2.1. D. Invalid Input for Command 1: Add a Fact

```
Enter the fact to add:
>>> [0] = [1]

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>>
```

Snippet 2.1. E. Valid Input for Command 1: Add a Fact

Facts can be perpetually appended to a database by repeatedly calling Command 1. As they are added, they appear as the final element in the list of facts, as viewable using Command 3: Show a List of Facts.

Command 2: Delete a Fact

Any facts that have been added can be singled out and deleted by using this method. This is best understood with a model. Let it be said that four facts have been appended in sequence using Command 1, then, the database state can be represented as:

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	…	Slot ∞
Fact A	Fact B	Fact C	Fact D			

Figure 2.1. F. Database after Four Fact Appendages

On such data, a console simulation as described in *Figure 2.1. G. Delete Slot Two Successfully* results in the new object state *Figure 2.1. H. Database after Four Fact Appendages, and Deletion*.

```
Enter the fact number to delete:
>>> 2
```

Snippet 2.1. G. Delete Slot Two Successfully

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	…	Slot ∞
Fact A	Fact C	Fact D				

Figure 2.1. H. Deletion of Slot 2 from *Figure 2.1. F.*

Notice that in *Figure 2.1. H.*, Fact B, which used to be in Slot 2, is now missing, and all of the facts that were to the right of Slot 2 have been shifted to the left by one. This demonstrates the behaviour of Command 2 when given a correct input, and it is left that error cases are to be explored, of which there are two; let it be presumed that the data in *Figure 2.1. H.* is what the program is operating on, :

```
Enter the fact number to delete:
>>> 4
Failed to delete fact: fact is out of range
```

Snippet 2.1. I. Deletion Range Error

```
Enter the fact number to delete:
>>> I've got mixed nuts and raisins, ;
Could not delete fact.
```

Snippet 2.1. J. Deletion "That's Not a Number" Error

In such cases that these errors occur, the particular message is important. *Snippet 2.1. I.* implies that there was likely an accident in selecting the index, and you should use Command 3 to check what fact they are trying to delete. The shown in *Snippet 2.1. J.* indicates that there is likely another issue.

It is best practice to always use Command 3 before carrying out any deletions, so that you don't accidentally delete the wrong fact and not realise.

Command 3: Show a List of Facts

The behaviour of this Command is somewhat intuitive, but it differs from all other Commands in that it doesn't prompt the user (>>>) for any additional information, and conversely, prints a new-line and then waits for a Command. Let a scenario be generated, and an example given, starting the program fresh:

```
Running PLE, welcome!
Press the return key twice to quit.

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 1
Enter the fact to add:
>>> [0] = [1]

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 1
Enter the fact to add:
>>> [1] = [2][3]

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 3
1 -> ([0])(1 - ([1])) + ([1])(1 - ([0]))
2 -> ([1])(1 - ([2][3])) + ([2][3])(1 - ([1]))
```

Snippet 2.1. K. Listing Two Facts

The two facts are listed in their respective positions, which is behaviour that agrees with the models in *Figures 2.1. F ., H.*. It can be noticed that the facts appear in their transposed form, namely, while they were inputted as A = B, they were outputted as the same mathematical relation, but equated to zero. The responsibility of matching input and output is entirely that of the user within the CLI, but, the GUI takes a more friendly approach (friendlier in general). The GUI is discussed in the next Part *(3. Using the Graphical User Interface)*.

The transformation between forms follows a simple mathematical process, demonstrated by example. Let the fact "[1] = [2][3]" be represented as "a = bc"

$a = bc$	The original equation.
$a - bc = 0$	Subtracting bc from both sides.
$(a - bc)^2 = 0^2$	Squaring both sides, because negative numbers can cancel with positives, causing miscalculations.
$a^2 - 2abc + b^2c^2 = 0$	Expanding the brackets.
$a - abc + bc - abc = 0$	Splitting the term -2abc. Also, in Boolean Algebra, a^2 can be changed to a, and b^2c^2 into bc etc.
$a(1 - bc) + bc(1 - a) = 0$	Factorising the expression. This is the form that appears printed.

Command Three has one other special behaviour that when either no facts have been entered, or all facts have been deleted, the following output is produced.

```
>>> 3
There are no facts.
```

Snippet 2.1. L. The Relativistic P.L.E - Response to Command 3 with an Empty Database

Command 4: Request a Solution

This command is the meat of the program. Requesting a solution is what causes all of the facts to be merged into one master fact, representing the whole complexity of the system, and then causes this information to be refined and returned by what the user is requesting.

The usage of this command isn't particularly special however. Starting with a new session, the following behaviour serves to be an example:

```
>>> 1
Enter the fact to add:
>>> [0] = [1][2]

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 4
Enter the request:
>>> [1]
0/0(1 - [0])(1 - [2])
1/0[0](1 - [2])
```

```
1[0][2]
```

Snippet 2.1. M. Requesting a Simple Solution

```
>>> 4
Enter the request:
>>> Salted cashews, ;
The request is malformed.
```

Snippet 2.1. N. Requesting a Malformed Solution, Causing an Error

Snippet 2.1. N. behaves quite logically, whereby a request failing adherence to *Part 4. Writing and Requesting Facts* causes an error message.

The earlier extract, *Snippet 2.1. M.*, is where mass calculations begin. The process of Command 4 is to create a master fact, and then filter it through a request. In the example, the master fact will simply be the fact in Slot 1, since it represents the entire knowledge of the system already. This fact is then algebraically manipulated according to the rules of (classical) Boolean Algebra, detailed in *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities - 1854 - George Boole.* Processing occurs such that the master fact is rearranged to have the request as the subject. Since the input was a fact, and the request a connection of symbols, and the mathematical processes to manipulate the facts all valid; then rearranging the fact to have the request as the subject yields the entire truth of the request.

Each line of output from this Command can be put into one of three categories: outputs with a coefficient 0/0, with a coefficient 1/0, and with the coefficient 1. The response of the system is the sum of each of these lines, in the case of *Snippet 2.1. M*, the equation that is sought after takes the form "[1] = 0/0(1 - [0])(1 - [2]) + 1/0[0](1 - [2]) + 1[0][2]", which, let it be represented as "a = 0/0b + 1/0c + d". This equation can then be broken down into two sub-equations for interpretation, with corresponding translations:

a = 0/0b + d	The class 'a' is comprised of some b's, and all d's.
0 = c	The connection of symbols c cannot exist within the universe of discourse.

The translation of the output of this Command is a similar process to the digitisation of facts, which appears in *Part 4.* of this manual.

Another scenario of output is also possible, namely, when there is not enough information to respond to the request, so the request (represented as 'a' above) is equated to 0. This results in no output from the Command.

3. Using the Graphical User Interface

3.1. Functionality Disparity between the GUI and CLI

Mastery of the GUI only comes from mastery of the CLI, at which point, the screen is designed to be intuitive and easy-to use. The user should have read and understood *Part 2. Using the Command Line Interface* before they proceed through this Part of the manual.

The GUI varies only slightly to the CLI in that it offers some additional functionality. The core differences come from the ability to place symbols in a Superposition State, where they are assumed to concurrently be true and false; and the ability to save and load databases.

Additional Feature 1: Superpositions

There is functionality in the GUI that allows symbols to be placed in Superposition. By example, let there exist symbols "Our Salted Peanuts" and "Famous Things of Ours", "Things Available for Selling", as well a fact symbolising "Our Salted Peanuts are Our Only Famous Things Available for Selling.". Let this fact be "a = bc", where the pronumerals are the respective symbols, then:

$a = bc$	The original fact.
$a(1 - bc) + b(1 - ac) = 0$	The same fact, transmuted and equated to 0, representing the combined knowledge of the system.

Setting the symbol 'b' to a Superposition, meaning that the system has no knowledge of what constitutes a 'b' (Famous Things of Ours). This introduces uncertainty in the system:

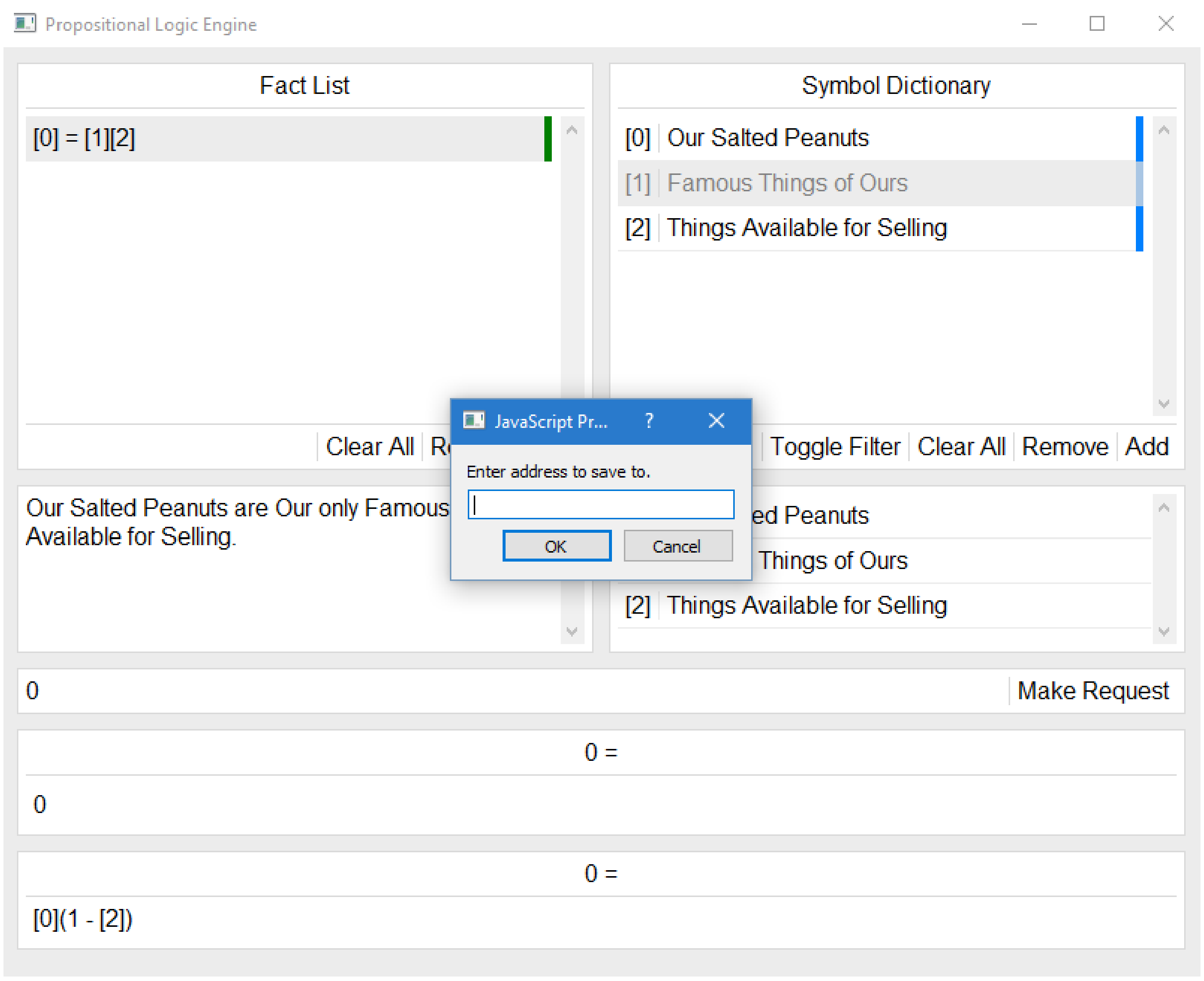
$a(1 - bc) + b(1 - ac) = 0$	The transmuted master fact.
$a(1 - 0c) + 0(1 - ac) = 0$	The fact after setting b to 0.
$a = 0$	(simplified)
$a(1 - 1c) + 1(1 - ac) = 0$	The original fact after setting b to 1.
$a(1 - c) + (1 - ac) = 0$	(simplified)
$a[a(1 - c) + (1 - ac)] = 0$	The product of the fact with b set to 0 and 1, meaning that b is now in Superposition.
$a(1 - c) = 0$	The simplified aggregate knowledge of the system, where all reference to b has been omitted.

This now represents what the Propositional Logic Engine knows about the symbols 'a' and 'b' ("Our Salted Peanuts" and "Our Famous Things"), that is, assuming it is not known what are the "Famous Things of Ours", it is still true that, on interpretation of the fact, "All of Our Salted Peanuts are Available for Selling".

This feature should be used when there is uncertainty within the system. It can be enacted as per the diagram in *Part 3.2. Screen Annotation*.

Additional Feature 2: Saving and Loading Databases

The shortcut keys Ctrl-S and Ctrl-O can be used to respectively save and load databases, as is standard throughout many programs. On calling a save or load operation, the user is prompted for an address to load or save from, both prompts functioning similarly:



Screenshot 3.1. A. The Save Prompt

From which, the user can press the escape key (Esc) or click Cancel or the pop-up's cross to resume normal functionality.

The user can then type a file path, for example, typing (without the quotation marks) "Save1" will cause the program to attempt to open or save to the file "Saves/Save1.ple".

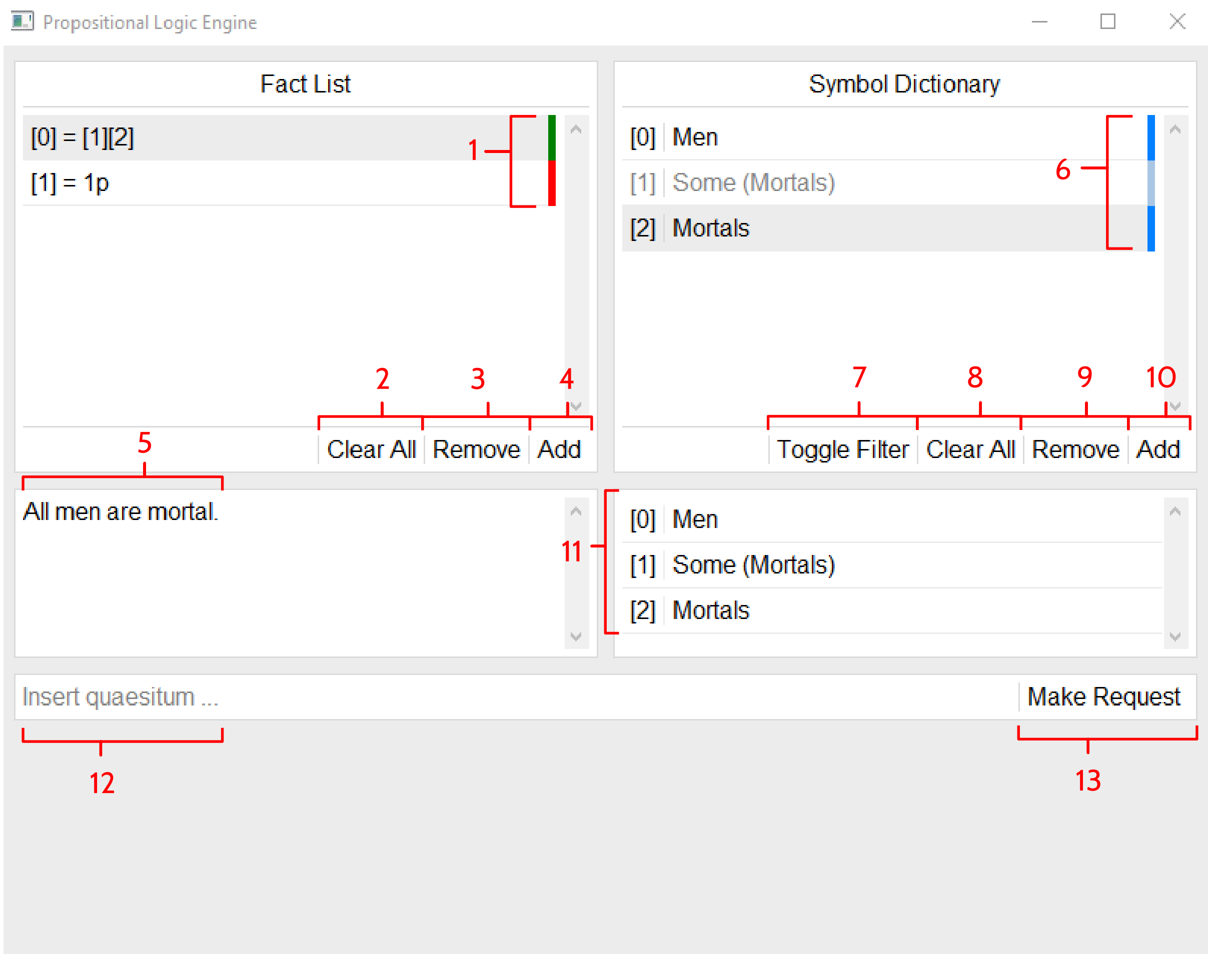
Directories can be navigated in a more advanced way with this command, for example, opening or saving to "/directory1/save2" and "/../../directory2/save3" are both perfectly valid expressions, provided that the directories already exist.

Attempting to load from an invalid directory or a non-existent file causes an alert box to display "Could not find file", but after which, functionality resumes normally. Attempting to save to an invalid directory gives a similar alert "The file path is invalid".

There is no warning when files are being overridden.

3.2. Screen Annotation

The GUI is designed to be intuitive, but an explicit explanation of the system may serve some ends. Two annotated screenshots follow.



Screenshot 3.2. A. PLE Before Use

1. Colour coding representing the validity of facts. Green if a fact is valid, and red if a fact is invalid. The colour orange may also appear to indicate that a fact contains too many symbols to verify the validity of, so, the responsibility of checking falls to the user.

2. Button to clear all of the facts. Clicking will cause a confirmation message to follow.

3. Button to remove the currently selected fact.

4. Button to add a new, empty fact to the bottom of the list.

5. The description for the currently selected fact.

6. Colour coding representing whether or not a symbol is in a Superposition. A lighter blue, accompanied by a greyed out number and name, confer that a symbol is in a Superposition.

7. Button to toggle whether or not the currently selected symbol is in a Superposition, and should thus be filtered from the output.

8. to clear all of the symbols. Clicking will cause a confirmation message to follow.

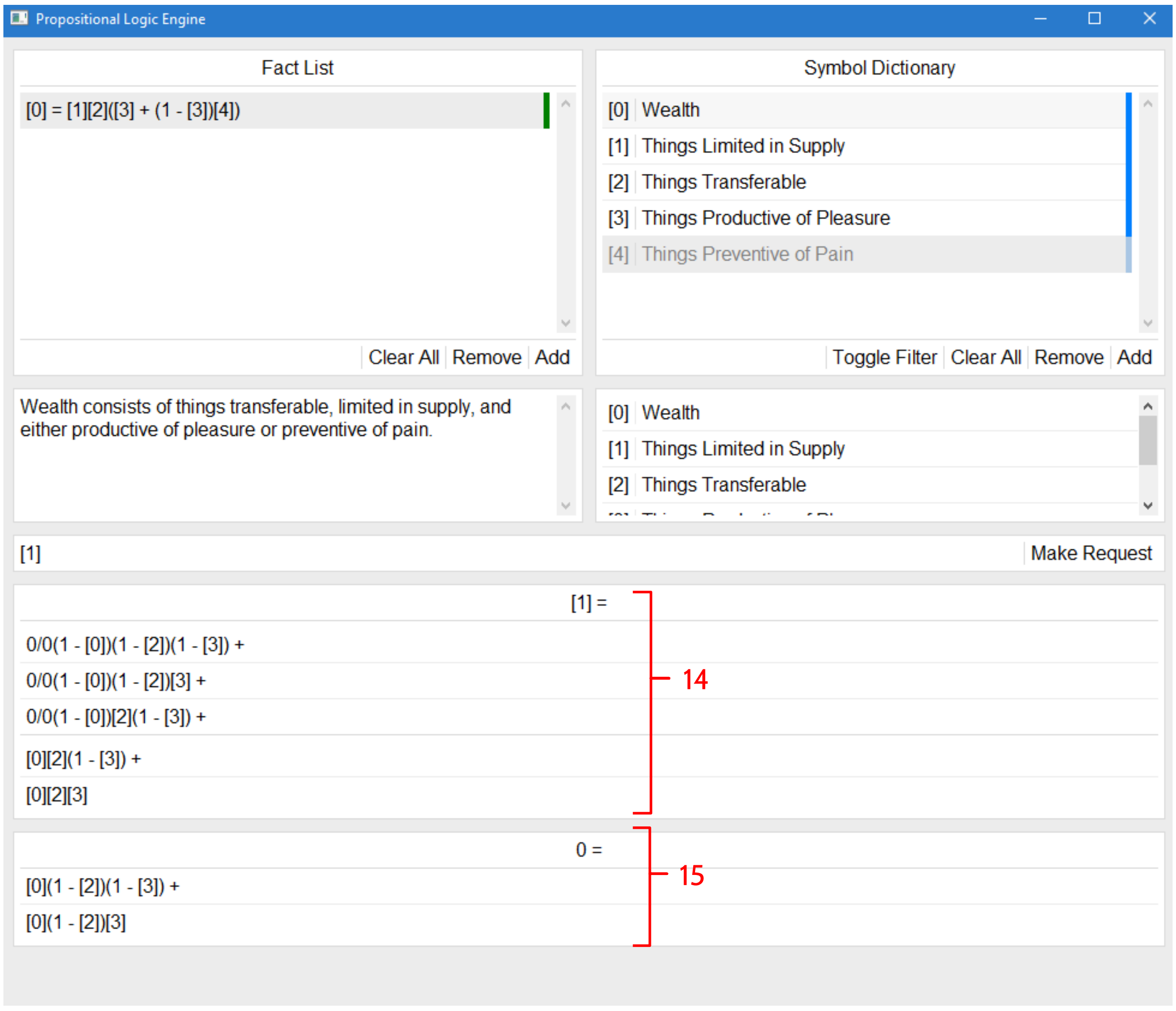
9. Button to remove the currently selected symbol.

10. Button to add a new, empty symbol to the bottom of the list.

11. The list of symbols that are present in the currently selected fact.

12. Textarea to insert a request (by analogy, the argument to Command 4 from *Section 2.1. Commands and Arguments*).

13. Button to launch the request, and begin processing.



```
[0] = [1][2]([3] + (1 - [3])[4])
```

[0]	Wealth
[1]	Things Limited in Supply
[2]	Things Transferable
[3]	Things Productive of Pleasure
[4]	Things Preventive of Pain

Wealth consists of things transferable, limited in supply, and either productive of pleasure or preventive of pain.

[0]	Wealth
[1]	Things Limited in Supply
[2]	Things Transferable

```
[1]
```

```
[1] =
0/0(1 - [0])(1 - [2])(1 - [3]) +
0/0(1 - [0])(1 - [2])[3] +
0/0(1 - [0])[2](1 - [3]) +
[0][2](1 - [3]) +
[0][2][3]
```

```
0 =
[0](1 - [2])(1 - [3]) +
[0](1 - [2])[3]
```

Screenshot 3.2. B. PLE After Use

14. The equation representing the response of the system, namely, the facts about the requested subject. (Divided into two subsections.)

15. Implied relations in the system of facts, that is, things that cannot possibly exist.

3.3. Zooming In and Out

Particularly, The Windows 8 and 10 operating systems provide "scaling" functionality, which is intended to allow the adjustment of text size on high pixel density displays, such as high resolution laptop screens. This can cause the PLE

In this release (v1.0.), there is no implemented way to change the text-size of the program without adjusting the screen resolution. In suchrepre a case that the software is rendered unusable by PySide's poor compatibility with this scaling functionality (or more likely the poor compatibility of the Windows OS feature), the level of zoom can be adjusted manually.

Step 1. Locate /PLE/PLE/GUI.html

Step 2. Open the file with a text editor, and locate the text (on line 22)

```
html
{
    -webkit-user-drag: none;
    -webkit-user-select: none;
    cursor: default;
    height: 100%;
    width: 100%;
    zoom: 1;
}
```

Step 3. Adjust the value corresponding with "zoom" according to what direction of zoom the user intends, and save the file. Values such as 0.85 and 1.5 are accepted.

Step 4. Restart the program.

4. Writing and Requesting Facts

4.1. The Digitisation of Facts

The digitisation of facts (and similarly, the interpretation of the solution) is the hardest process in using the system. This section by no means trains you in using the system wholly, but merely introduces you to Boolean Algebra in a simple and understandable way.

- Facts are represented as connections of symbols, eg., [0] = [1] implies that [0]'s and [1]'s are overlapping sets.

- 0 = [0] implies that [0]'s don't exist.

- 1 = [0] implies that everything in the universe is a [0].

- The selector [0][1], conferring multiplication, refers to "[0] and [1]". Thus, 0 = [0][1] implies that no object in the universe concurrently possesses the properties [0] and [1], or rather, nothing at once belongs to the class of things represented by [0] and that of [1].

- (1 - [0]) refers to not [0], so (1 - [0]) = [1] means that if an object possess property [1], it will not have property [0], and likewise, if the object possesses property [0] it will not have the property [1] etc.

- '0/0' is indicative of a symbol in superposition, so by 0/0[0] is meant the same as [1][0] if the symbol [1] is in a superposition. This symbol is best understood as representing "an indeterminate amount of things: some none or all", so, the fact [0] = 0/0[1] can be read, for example, as "All [0]'s are [1]'s", or "nothing that is a [0] can not be a [1]". It may help to think as [1] as a superset, and [0] as the subset.

- The addition sign + means "or", so [0][1] + (1 - [0])[2] refers to things possessing either the joint properties [0] and [1], or property [2] in the absence of property [0]. It is important to recognise that addition can only be performed between mutually exclusive sets, and the PLE will pick up on this if such an error as [0] + [1] is detected.

It may be best that some examples are studied, as referenced in *Part 5.*.

4.2. Formatting Rules

The PLE expects that only single characters of whitespace are adjacent to operators +, -, (,), or =. Apart from this, any mathematically valid Boolean Expression is accepted by the program. EBNF for the accepted format does not appear here, due to this simple reflection of the mathematical system.

5. Examples

5.1. Sample Solutions

The solution package comes with three sample scenarios in the Saves directory, which can be opened by pressing Ctrl-O. They are named "All the animals in the yard gnaw bones.ple", "I always avoid a kangaroo.ple", and "No badger can guess a conundrum.ple" (each openable by trying the name without the file extension).These are three sample models of some Propositional Logic examples as authored by Lewis Carroll.

5.2. Sample Problems

Engineered by Lewis Carroll, "Sample Problems.txt" appears in the solution. This contains a list of examples of problems to which the PLE can be applied, although, each is entirely nonsensical. This domain of example does not render this system of truth any less viable to more serious applications.