SOFTWARE DESIGN AND DEVELOPMENT
MAJOR WORK

NICHOLAS KILLEEN, 2016

CONTENTS

I. PROJECT MANAGEMENT

II. DEFINING & UNDERSTANDING THE PROBLEM

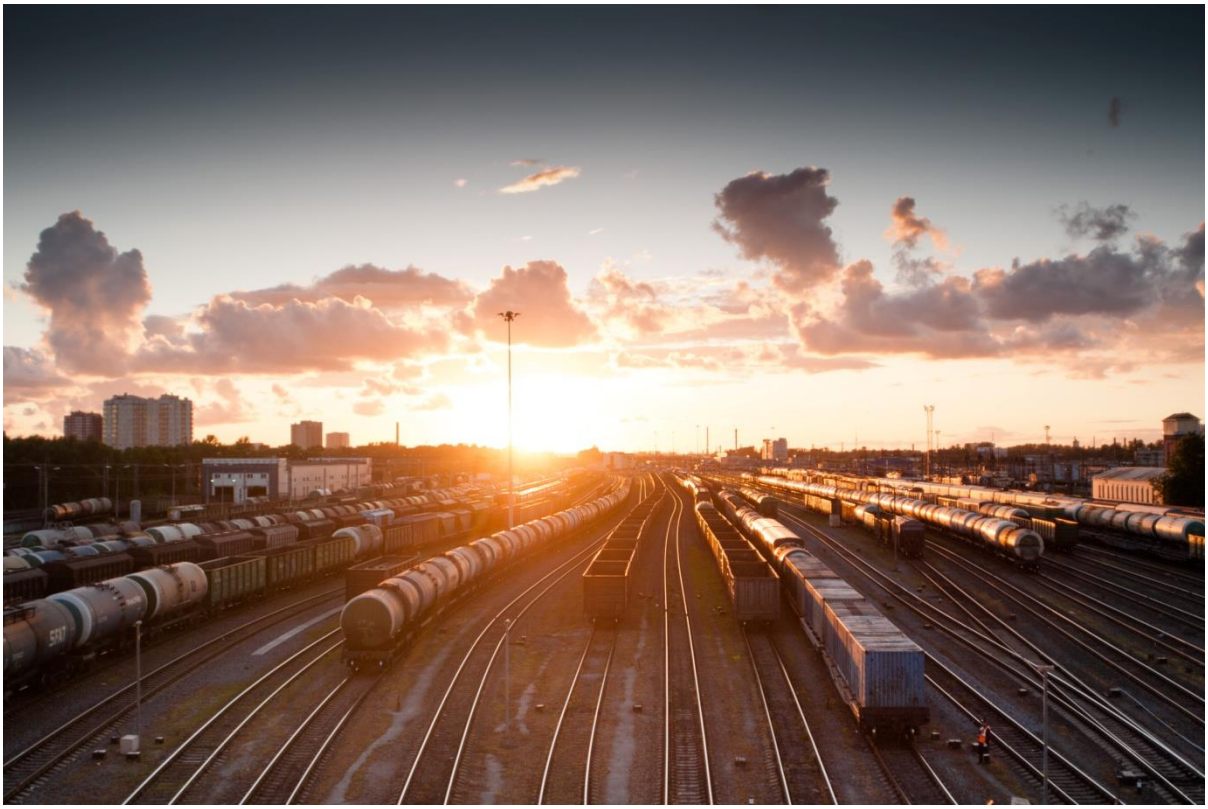
III. PLANNING & DESIGNING THE SOLUTION

IV. IMPLEMENTATION

V. TESTING & EVALUATING

PROJECT MANAGEMENT

PART I.



1. LOGBOOK

1.1. DIVINING THE PROBLEM
[08/04/2016]

Situation prevents the execution of usual procedure when it comes to selecting a problem to solve. In the purest case of problem solving, the problem that is perceived to benefit most from solution is the one that is targeted to be solved. The process of problem solving by its very nature is natural and responsive. The goal of the act "problem solving" is to actually solve a problem to a certain degree of quality, as necessity dictates.

Software design and development is a specific instance of problem solving, and it ought to whence inherit the same honest purpose. In contradiction I must remark, although the header of each page is 'Software Design and Development", this report isn't one of software design and development because it isn't one of natural problem solving, simply because it does not work wholly towards the end of selecting and solving the problem that would benefit most from solution.

This assignment, by its form, is not one of software design and development since it is not one of problem solving. By my adherence to what is stipulated in the cover sheet, it has running parallel to the end "to solve a problem" the primary goal "to produce a brilliant report and wonderful documentation such that an assessor deems my execution of the art of problem solving to observe high quality". To this corrupted end, the process of selecting the most appropriate problem to be solved quickly turns into the act of "divining" a problem -- choices are no longer dictated by necessity of solution, rather, a problem must be invented or skewed so as to allow desirable demonstration of aptitude. The problem, here, must conform to the predetermined nature of solution, rather than the solution having to adapt to the problem. I think this would be a mild trouble on its own: to begin the project, perhaps a moderate amount of thought must necessarily be put into divining the appropriate problem, -- but there are further restrictions in that "you are required to find a *real* client".

Birthed two years before me I have a brother. Luck permits him to be in a situation where a software solution is honestly appropriate to a problem which, luck further has it, allows the showcasing of project management, design, and development practices, as well as general aptitude in line with the strict criteria. The problem is concerned with the theory of probabilities, and it is truly great luck that this problem fits every desirable epithet explored hereto. Prior to my awareness of the timely existence of this problem, I had begun divining an issue appropriate to the criteria, and I chose the subject of propositional logic. Quite coincidentally, propositional logic and probabilities are close neighbours, probabilities being an extension of binary logic. I have experience in propositional logic, but am uninitiated in probabilities, so it follows the two potential problems have their advantages and disadvantages. The question is now: which of the two problems should I solve -- what is the more favourable of the two sets of circumstances:

1. To solve the problem of propositional logic. To begin instantly on a topic I have experience with. To make a product of perhaps only little use to the client. To have to accentuate the need for a solution in a non-straightforward manner.

2. To solve the problem of probabilities. To read a most terse text just to analyse the feasibility of a solution (George Boole's *An Investigation of the Laws of Thought*), so as eventually to educate myself on the general methods in probabilities and attempt to reach a computerised solution -- which is a process of indeterminate difficulty. To solve a real world problem that has been naturally encountered, and not divined.

This, it now appears, is a question of integrity. The specification of a need for a real client, which I project to be present in attempt to force a perception of integrity, leads eventually to the dilemma it was in place to prevent. A third problem option is unlikely to spontaneously appear, but I would very much welcome it. It is important, for the sake of imperative progress, that a plan is made; yet, the hasty decision between the two would be equally disastrous in the worst case, regardless of the choice. With the blessing of inexperience in the domain of moral dilemmas (or perhaps just ignorance), I can quickly overlook many problems that may be clearly visible to others so as to reach a conclusion.

For one week I shall read patiently into the theory of probabilities. I may then come to realise that time may not permit me to learn the theory of probabilities to the degree of sufficiency necessary, and this is very much the more likely case, much to my chagrin: in the scenario of equal difficulty, the problem of probabilities would be more useful and enjoyable to solve. I become cognisant of the lack of feasibility or practicality this problem may be characterised by, it will be abandoned and, possibly harshly, never reconsidered. If I learn enough within the week to have confidence that the harder problem is quite doable, I will consult the client (consultation has already occurred extremely informally) and begin working towards a solution.

Optimally, questions of probability will quickly be found generalisable in a way that is similar to questions of logic, and this does appear to be the way that things will go if I have more time. Perhaps within three weeks I would have confidence that the idea is workable, but this will never be revealed because of the self-imposed deadline of a week. This deadline is however necessary so as to facilitate some sort of progress if the concept of probabilities becomes unworkable, which would otherwise surely be recognised too late.

1.2. PROBLEMS WITH URNS
[09/04/2016]

I observe a commonplace habit of reading books from top to bottom, and standards do indeed permit this to be the most successful strategy in most readings. Boole's *The Laws of Thought* is, as has already been expressed, ridiculously pithy -- to an extent that, to consume the knowledge of the first logical half, two years of reading have been spent (albeit undisciplined reading). It is certainly not unreasonable that, two years after reading a note in the preface, I no longer remember what the note was referencing. For much of this day's readings, I have reflected in a state of perpetual perplexity on

an exemplification of Boole's second proposition under *XVII. General Method in Probabilities.* Quite ordinarily, I was confused by the philosophical explanation of the rule as it had not been tied town to any specific examples thereto, and I accepted this state of confusion as is purely necessary in the comprehension of this work (this is an example of forward declaration). I looked then to the example, which I followed through several times only to feel more confused. Eventually I simulated the example, which involved the drawing of balls from an urn, but this just led to what I believed to be further mathematical contradiction. Scepticism of an error in the scanning fuelled my continued exploration of the proposition -- Project Gutenberg is wont to observe extremely mild analogue-digital transmutation errors that sometimes warrant correction, and this, I thought, may be an extreme case.

After much annoyance, where the results seemed to be entirely uninterpretable, I did a simple Google search of the first sentence of the question. This yielded a critical mathematical exposition by T. Hailperin, in which a reference was made to a note at the beginning of the book -- which I quickly made recognition of, and hastily read. Comically, the preface of *The Laws of Thought* has as a portion the following, in reference to the proposition in question:

> "I think it right to add, that it was in the above form [corrected form] that the principle first presented itself to my mind, and that it is thus that I have always understood it, the error in the particular problem referred to having arisen from inadvertence in the choice of a material illustration.", --

Misunderstandings such as this will undoubtedly continue to slow the process of reading to a degree that is probably well beyond recoverability. By even generous extrapolation, I am very much in doubt that I will have enough haste in learning to be able to solve problems of probability.

1.3. SELECTING A LANGUAGE
[10/04/2016]

Prior to the official commencement of the project, much informal and unlogged research was undertaken regarding what languages easily synergise GUI and file IO. I pity anyone who requires networking on top of these requirements (with blessing, these peers only exist theoretically in hyperspace). Ideally (and in line with what I learn in this course), the selection of a language is able to be delayed until the implementation stage. This cannot work in this scenario, firstly, because it must be identified whether the problem is one (for simplicity) that warrants an object oriented or procedural approach.

The problem -- regardless of whether it is one of logic or probability (logic continues to seem more likely) -- is of a strange type. Indeed, the solution is simply a model of the problem from which some data can be found, and this is conducive of an object oriented approach, and whence, languages I would consider using would include C#, Java, or the "++" half of C++. The solution to the problem, however, is a general method; it is able to be represented entirely procedurally without much complexity at all, and languages such as C (or the "C" half of C++), Python, or JavaScript would be also appropriate. The set of languages that the program is to conform to changes both the nature of

planning, and how the problem is to be represented in the "defining and understanding" phase, thus, it must be given immediate consideration.

As is, the division remains relatively simple, and no specific language has to be chosen -- just a set of languages. The solution however is necessarily divided into two components: functionality components, and human interfacing components, which confuses things somewhat. In elimination, I decline to use C# because it is "too object-oriented", that is, all actuation is based on real time events, and this feature is not pertinent to the problem at hand. I feel that Java would certainly be optimal, however, the requirement of GUI becomes an issue when it is remembered that the functionality provided through the standard Swing or JavaFX libraries is quite limited to what is perhaps needed in the program unless I am to endeavour to learn the intricacies of these libraries. I can't use C or C++ easily due to the complexity and rigidity of the libraries, meaning that there is a quite a learning curve in GUI programming with C (in approximate parity with Java). For Python GUI, there are no libraries which immediately stand out. JavaScript, due to its friendliness with CSS and HTML/XML, would be perfect for GUI, -- however, it lacks the requisite functionality of file IO (file IO is achievable with browser extensions or the actual hosting of a server). The DOM that is seen in HTML/XML, CSS, and JavaScript is infinitely malleable and is really what every GUI system should aspire to be like (although CSS can be extremely hacky).

After some research, https://wiki.python.org/moin/GuiProgramming yielded that there exists a python library which creates a DOM, and within which, JavaScript and its adjacencies can be executed; PyQt and PySide both offer functional DOM renderings. This means that the GUI can be considered entirely separately from the actual methodical solution, and much of the defining and planning stage can be actuated in utter ignorance that there will be a GUI at all. It may be noticed in section `I. 2.1. GANTT CHART that the overflow between the tasks *Programming core* and *Implementing UI* is only 12% of the combined time span of the tasks, and it is worth noticing that this is a result of this language specification as well as the affects that the choice has on the earlier stages.

…

1.4. ABANDONMENT OF THE GENERAL METHOD OF PROBABILITIES [15/04/2016]

For the past several days, research of probabilities has been fruitless, and my current level of knowledge and rate of success is not conducive to the success of this nature of project. One week has passed, so by my past decision, all work on the general method of probabilities will be cancelled, and instead, a problem is to be found in the domain of propositional logic. The topic of probabilities has been disbanded not because of its lack of learnability, and not due to the perplexity of the topic, but rather, due the time which it takes to learn.

I have conducted preliminary investigations into the applications of propositional logic to Law (so as to divine a problem), informally querying my brother about the usefulness of argument assistance

in many aspects of Law, and pragmatism has ensured its place as the problem to be solved. With this conflict resolved, the project can formally begin.

1.5. IDENTIFYING A NEED
[16/04/2016]

I interfaced with my brother, partially via Facebook, so as to write section `II. 1.1. on "asserting the need for a propositional logic engine" after he shared manifold relevant resources from *The Oxford Companion to the High Court of Australia*, *The New Oxford Companion to Law*, and Stig Kanger's *Law and Logic*. The creation of this section presented no specific difficulty: there is a real, effable need, upon which can be expounded a more descriptive account of what exact problem is targeted for solution. It is merely required that I pose the need as being imperative, when really, it could be done without, but it is a need that does exist.

Following this first section of the problem specification, I had anticipated a title *Objectives & Values*. It is important that the objectives of the development are made explicit, but verily, it has come to my realisation that the objective is simply to squash the need sufficiently, that is to say, to assist in the application of propositional logic to any matter of argument. The objectives are not especially obfuscated or interpretable, and I believe they do not deserve any mention as they have already amply been conferred in the section on needs, albeit tacitly. The objective of the project is to meet the need, and this needs only to be elaborated upon when the requirement report is created. Under the heading, I initially put "to have a system that assists in the application of propositional logic to any matter of argument", but this has since been flagged as superfluous, leading to the removal of the whole heading.

1.6. DEFINING PROCESSES
[17/04/2016]

In a time prior to today, it has become revealed to me that I have still not explicitly defined the problem, and therefore, corrective action needs to be observed. The specification (which will hereafter be reworked) does not describe the logical process that the program's functionality will be an analog to. The solution to this, in an optimal scenario, would be a creation of an input/output diagram. Were one included, all ambiguity about what transformation occurs would be removed -- but, if this section is looked upon, the nonexistence of one will quickly become obvious. There is, however (now) an IPO chart, where convention separates the input and output by a corruptive process. What purpose does the process serve?-- in the defining stage, there should be naught concern regarding the abstractions of a process; the definition of the problem is only to get from inputs to outputs. To define how this black-box transformation is actuated so early on in the development lifecycle is to force the development to be centred around that surely ill-thought of plan. The specification should not care how a solution is reached, and this, I believe, ought to be made a consideration of proponents of early IPO charts in large development scenarios. My beliefs, conversely, must be contradicted in that it is specified that sufficient details are to be provided in the *process* column. This unfortunate opposition leads the contents of this column to be somewhat ambiguous, and the actual process remains to be defined how dedicated planning sees fit.

1.7. FEASIBILITY
[18/04/2016]

I began writing the feasibility study quite sceptically and speculatively. My concerns are encapsulated mostly therein (`II. 3.), and warrant no further mention wherefore. Importantly however, I began the process of backing up the project and associated files. Hereafter, one online backup will be made each day any files pertinent to this report are non-trivially improved. The importance of this is manifest, and needn't be elaborated upon unnecessarily

…

1.8. SURVEYING
[21/04/2016]

In my two days of absence, my efforts can be best described as "refactoring", as well as some additions being made. As for why this is undocumented in the logbook, it is because I let these days be of rest, and it was not expected that any work was to be done on these days. The additions were optional, meaning they do not warrant the necessity of proof of occurrence. All that was added, really, was some details of a survey, which is frankly too stressful and depressing to be commented on. I feel I must justify its short length: it only asks for information that is exactly relevant; the goal is not to identify the target market (which would be the purpose if development were to occur at a full scale, but a much wider audience must then be exposed to the questionnaire); the goal is, rather, to establish the fact that there is a market at all. There are many other things (in terms of validity of surveys) worth acknowledgement, but they should be obvious concerns of anyone who creates a survey, not particularly me, and escape this discourse thus.

1.9. THE NON-EXISTENCE OF A PRESENT SOLUTION
[22/04/2016]

The title of this entry was planned to be shared in the specification, but I have made attempt to consolidate subjects of encroaching argument to the logbook, and any sort of repetition is somewhat superfluous. Truth tables, which covers much of the functionality that the P.L.E will have, do not sit well with quantifiers or qualifiers or other epithets to this effect. -- here is why truth tables are not up to scratch to problems as extreme as those in Law (and particularly, why they annoy me):

- They are physically quite bulky

- Tools to generate truth tables are not easily usable (in terms of higher level functionality).

- The process of translation and interpretation in truth tables, in my opinion, is much harder than that in real Boolean algebra.

- Truth tables are not especially versatile (all manner of questions can be answered much more easily with Boolean algebra than with a truth table) -- truth tables are made to verify conclusions, not to reach conclusions.

- Truth tables are designed, primarily, to handle binary states -- pure Boolean algebra is the quantum computer to the computer of truth tables. Despite the connotations of the term "Boole", Boolean algebra allows the states 0, 1, and the quantum state n = 0/0, where n(1 - n) = 0.

- It is difficult to store results of truth table calculations for future use.

- Truth tables are intended for mathematical problems, not problems of broader argument.

Further, all online resources for truth tables are simply lacking, and I cannot understand why. I thus remain fairly uneducated in making some of these assertions -- but I have conducted plenty of research, and any statement of lack of functionality, if invalid, still confers the truth that the utilities provided to perform a certain function are unclear and terribly hacky.

As for tangible processes actuated today, it can be said that the feasibility report was refined and completed, and I have begun titling headings under chapter `III.

1.10. A MATHEMATICAL DEFINITION OF THE PROBLEM [23/04/2016]

I have began to analyse the problem purely mathematically, but the working has quickly become so complex that I am unable to continue. Tomorrow, I must rewrite the proof in a more followable way, and try to restrict the use of symbols. The actual mathematics isn't by its nature complex, it is only so by this being my first attempt at a proof -- after iterations, it will surely become of higher quality.

1.11. META-META-BOOLEAN ALGEBRA [24/04/2016]

I refined and rewrote the mathematical analysis just once, and it is of a much improved standard, and will undoubtedly help in the planning of a solution. Quite funnily though, I reflected upon the actuation of a similar task in past projects, and realised that -- regardless of the problem -- I would have used likely have used Boolean algebra to help define the problem, or perhaps to analyse an algorithm (an algorithm has not yet been written yet, so it is possible that such analysis will be seen under this project). In brevity, I have been using Boolean algebra to define a problem that involves the analysis of Boolean algebra.

This perhaps demonstrates the applications that components of a P.L.E will have in software design and development, although it would require less derived interfacing controls for the more technical users. I must therefore assert that the technical manual is to a high degree of quality, as is the functionality of non-human interfaces. (In appendage 30/07/2016, the technical manual is obsolete.)

These claims must be instantiated, so I make citation of a past work, *Egg Classification Report for NSW Egg Board*, under the title *Deskchecking Logs*, Boolean algebra is used to prove the correctness of a subroutine. It must be recognised that the purpose of the citation is solely for exemplification of the existence of a problem, not for the convenience of reusing what has already been said in making a similar point. I began with the premise:

"Satisfactory points consist of those in unbounded ranges, or those within the boundary. Points within the boundary can be either above the point of exclusion, or lie on the exclusion point where the boundary allows it."

I proceeded to assign the symbols 's', 'u', 'a', 'o', 'e' to the objects involved in the proposition, and used a shoddy JavaScript tool to prove that the Boolean expressions are equal:

$0 = u + (a + oe[1 - a])(1 - u) - s,$
$0 = a + u(1 - a) + oe(1 - u)(1 - a) - s,$

The JavaScript tool I used was included in the appendix, and the functionality of which is hard not to include in the program:

```
function writeRow(s, u, a, o, e)
{
	equation1 = u + (a + o*e*(1 - a))*(1 - u) - s;
	equation2 = a + u*(1 - a) + o*e*(1 - u)*(1 - a) - s;
	LHS = Math.pow(equation1, 2);
	RHS = Math.pow(equation2, 2);
	disparity = Math.abs(LHS - RHS);
	document.write(s + "" + u + "" + a + "" + o + "" + e + "" + LHS +
	"" + RHS + "" + disparity + "<br>");
}

for (i = 0; i < 31; ++i)
{
	number = i;
	s = number % 16 != number;
	number %= 16;
	u = number % 8 != number;
	number %= 8;
	a = number % 4 != number;
	number %= 4;
	o = number % 2 != number;
	number %= 2;
	e = number % 1 != number;

	// multiplying the arguments by one converts them from
	// booleans into integers implicitly
	writeRow(s * 1, u * 1, a * 1, o * 1, e * 1);
}
```

Were I here not to have used Boolean algebra, I would have had to have done a maximum of 64 additional manual program run-throughs. I echo my complaint:

"As a person, I am not at all happy to manually evaluate two different quinary equations a sum of 64 times. Surely, optimisations can be found (some quite obvious), but, instead, as resources allowed, the JavaScript snippet was created and used to construct the constituent map."

It is obvious from this citation that lower level Boolean algebra is useful in software engineering, and this is further observable in the first mathematical analysis `III. 1.1., although it is difficult to discern between Boolean algebra and meta-Boolean algebra -- so much so that I don't know which means what. Surely, Boolean algebra will be used again in the development, and perhaps it will warrant a callout.

1.12. ON SYSTEM FLOWCHARTS AND THEIR REDUNDANCY [25/04/2016]

Today I wrote the first system flowchart in 'III. 1.2.. Verily, the system is of such a simple nature that any system flowchart is utterly useless. I already knew -- and all with any power of inference ought also to know -- that the system has a database, and receives inputs and outputs from the operator in a digitised and local sense. It's a waste of space.

In examining the flowchart, firstly may be noticed the lack of symbol diversity. There are only processes, inputs/outputs, and a disk drive. Were the problem more suited to a system flowchart, a wider variety of symbols would be observed, not just those which can be perfectly encapsulated within a DFD. The problem simply isn't suited to this type of model, and its sub-modules are even less so compatible. Why is this? When a fact is deposited, it is done so instantaneously, and then the system idles. When a fact is requested, it is done so instantaneously, and then the system idles. -- Yes, even if the process takes upward of nine thousand years, the computer never waits on any further input from the outside world. System flowcharts are designed to model the logic and data-interrelationships between modules and the outside world, but there simply is no relationship here, and there is little I can do to feign it.

Even if the system was depicted to be using multithreading, there would still be no real idling from the perspective of the outside world. Each CPU simply dumps its result into a database, but instead of idling in the case of earliness or lateness, they simply retire. They don't sit and wait for a manual operation, or an online input -- they finish their process and forget about everything they have done, until they are perhaps called again to an entirely new problem. The whole process, from the one manual start to its oblivious end, is entirely automatic, and any manual interruptions are either usual terminations or pauses. The P.L.E, in best summary, doesn't behave extrinsically like a system since it has a very simple input-output interface, and it is thus that it may be clearly perceptible a lack of ease in creating the first flowchart.

For the past 10 or so days, I have not failed to work on the project to some extent. For the week before that, I was reading intently into a book directly related to this project. I feel that the efficiency of my work is suffering in correlation. This may very well be a fact of life, and there was an initial spike of enthusiasm, or perhaps I have simply been working through tasks of appreciating "toughness". Mayhap it follows from the length of the project: to go from one to two pages is to double all work done, but to add one page to thirty feels menial. Irrespective of cause, I perceive that my efficiency is declining, and to hold such a perception is to assert its truth, and in wise correlation, I have chosen to take a short break.

…

1.13. (MORE) BOOLEAN ALGEBRA AND THE REAL WORLD
[28/04/2016]

Athwart the gap in continuity of work, I have observed more Boolean algebra in the real world. This isn't relevant to the external properties of the P.LE such as potential inputs and outputs (the behaviour is still able to be modelled, but it would be for no sane end), but rather, it is pertinent to demonstrate the commonness of how the solution is implemented, and indeed the general problem.

After receiving results from examinations, I added some functionality to an Excel spreadsheet markbook which I had created in preparation for the year. I noticed a column in a subject table labelled "*enum*" (short for enumerable), particularly referencing a Boolean answer (given its context) "Is this score enumerable", that is to say, "has the student sat this exam?". Because it is difficult to implement complex conditionals in Excel, exploration revealed I used this enum column to facilitate basic Boolean calculations. Enum was defined as:

```
IF (TEST_SCORE <> 0, 1, 0)
```

And an average percentage was calculated by summing the percentages, and dividing by the sum of the enum column. A similar thing was done to calculate the WAM.

```
averagePercentage = SUM(PERCENTAGE_RANGE) / SUM(ENUM_RANGE)
```

This serves to exemplify further (I acknowledge surely unnecessarily) the permeation of Boolean algebra throughout areas of maths and computer / software engineering, which can hopefully be generalised to "Boolean algebra is useful". With further haphazardness and some very generous assumptions, I can furnish the point that not many people know about Boolean algebra -- although I simply wish to say "peradventure, not many people know about Boolean algebra" in a much more modal tone of reading. It so happens that the Excel spreadsheet was an instrument of collaboration, and that the collaborator (lacking a channel of communication with me) ignored the manifest functionality of the enum column in favour of re-implementing a portion of the spreadsheet so as to add a feature. It would have been much more convenient had they known Boolean algebra.

…

1.14. MICRO MANAGEMENT
[01/05/2016]

Today (and also yesterday) was spend mindlessly designing a storyboard in futile detail. The actual GUI will surely be quite different to this plan, but it is certainly a worthy endeavour to design a draft, especially to submit to the client. The storyboard can also be an early review model for feedback just before the actual GUI is developed. It was shared with the client via Facebook (more formal methods would be appropriate under a discrepant set of circumstances), who was contented with the present functionality for the moment.

As I began writing the GUI, I had mind to create a working model, but realised quite quickly that it would be a ridiculously excessive task. The initial storyboard is only meant to be a vague sketch of concept, and does not need to be detailed or interactive, for the structured approach in no way mimics the prototyping methodology. I thus made additional effort to do a fairly shoddy job: there is a lot of noticeable whitespace, poor alignment (which will definitely not be seen in the implementation), and the inner properties of each screen are left abstract.

1.15. DATA DICTIONARY
[02/05/2016]

Quite mundanely, I wrote the data dictionary (`III. 3.1.) without much challenge, but did notice that most of the types were abstract. It was difficult to decide whether to do the data dictionary or function signatures first, since each are quite interrelated. Having chosen to do the data dictionary first means that functionality of many data structures will not be obvious to a reader until later, but this very correctly mirrors the actual situation. Each of these abstract data types can be implemented in a variety of different ways while still being correct -- so why should they be predefined when they can be experimented with later, and even in the implementation stage (with thanks to modularisation and well defined interfaces without side effects) data structures and algorithms can be torn out and replaced with alternatives easily.

I noticed that the data flow diagram in section `III. 1.2. omits filter functionality. I have written a note to iterate upon this diagram in the future, namely, when compiling the technical manual, but my time management strategies (I. 2.2.) expound that rather than planning to do things, they should actually be done. I here, go against my own advice, perhaps by laziness, or by some state which I am too lazy to detail (a wily observer will notice that, just then, to question my own laziness was to assert its existence in an act of lazy self-deprecation).

(An appended note): the explanation of filter functionality is trivial -- but let there be a mathematical summary:

> To eliminate symbol θ_n from a master relation, $V(\theta_n) = \delta(\theta_0, \theta_1, \ldots, \theta_n, \ldots) = 0$;
> Calculate V(0) and V(1), and set V to the result.

1.16. ESTIMATIONS
[03/05/2016]

Workings have been unchallenged and straightforward: I have finished the data dictionary, adorned it somewhat, and began on some more maths. I am only struggling as to whether I should develop my definition of information with reference to the predefined symbols when maths was last done, but I ought to easily see through this problem when I endeavour to work on the maths next.

The lack of challenges have brought room for reflection, namely, on time estimations. The planned and actual Gantt charts have been quite discrepant, but not unexpectedly; I did indeed expect a discrepancy, for the Gantt chart is planned for the worst case, but I felt I had achieved a realistic ratio

between tasks. Conversely, I significantly underestimated the time it would take to generate a storyboard -- which I believe is an error I have also made in a past project (pertaining to the weighing of eggs, which was quite an imperative project). Because of my apparent (obvious) habit of terrible estimation, I am sure that I will be surprised at my time allocations as I progress through the project. I much look forward to it.

1.17. FEATURE DEPRECATION IN CLIENT CONSULATION [04/05/2016]

While prototyping the interface, I realised that I must store in some form the result of division of two relations. This form is mathematically incomplete (or is commonly accepted as such -- I would prefer to attach the epithet of "deterministically incomplete"), and as a result, it is very difficult to apply mathematical transformations to the cumulated object in a digital environment. I realised that, during the creation of the storyboard (`III 2.1.), I made the assumption that filters can be applied to this form using the standard algebraic method: the elimination of x from an expression 0 = f(x, y, …), is 0 = f(0, y, …)·f(1, y, …);-- but this requires a non-mathematical implementation of the multiplication operator, which will surely prove difficult and perhaps unachievable.

A filter can still however be applied to the numerator and the denominator of the expression individually, but the development (by my unfounded projections) will still be exponential in size. I informed my brother (the client) via Facebook that the functionality of applying a filter after a quaesitum has been reached may not necessarily be present in the solution at a sub-exponential speed, despite what may have been interpreted by the storyboard [which I shared with him earlier]. He remarked in recognition, disappointment, and acceptance. His exact words were "I recognise this; am disappointed by it,-- but accept it."

The speed at which a filter can be applied to a finished result will be an important point of mathematical analysis in the future. This feature will be the one most subject to change and iteration.

…

1.18. CASE TOOLS [07/05/2016]

Today I made a giant structure chart. Structure charts are quite well known for being ridiculously large in proportion to the size of the project, and this was no exception. Truly, it was developed with much oversight by necessity of simplicity, but it still took the day to create.

I looked into available case tools to create structure charts, but the better alternatives were not free, so I decided to iterate upon drafts on paper, then implement the final result into Word. I think it wasn't the lack of CASE tools which made this a difficult task, it was the fact that structure charts aren't suited to many large solutions. A call stack, I propose, would have been a far better alternative, and a radial structure chart would have been worth trying (it is mostly the constriction of space that makes structure charts difficult to use).

1.19. CODE REUSE
[08/05/2016]

Serendipitous aspect (in general) has it that I have developed a mathematical eval function in C++ which could be reused in the solution, even if it is written in Python (for extensions can be added via dynamic link libraries, or the code can undergo mild translation). I am leaving this opportunity open in the future, but it will not be until I do the preliminary operational analysis before a decision is close to being reached.

…

1.20. HIATUS
[24/05/2016]

At this point in time, I am well behind my worst case Gantt chart, and thus, if I continue at the current rate, I will surely fail. It perhaps must be accentuated that almost two weeks have passed. My absence can be attributed to poor management of a multitude of other concerns, and my lack of willingness to resume at this point of the project (I stopped at an inopportune point, on a fairly depressing nature of task, for I cannot conceive of a time in which I especially enjoy writing pseudocode). I have created a simple checklist (although of 22 components) with tasks that can be completed individually. This combats the mindset "though I have some short period of time to work on the project, I will not, since that there is a large overhead to project work and a corresponding logbook entry must be annoyingly created" -- instead, due to the paper form of the checklist, small contributions can be made at arbitrary intervals, and every now and then, the digital version can be updated to reflect any changes.

For some time hereafter, logbook entries will be sparse or perhaps observably non-existent. In abandoning this responsibility which aims to assert productivity, auspiciousness is generated in productivity itself: I will no longer have to spend as much time logging the things I do as actually doing them, thus discouraging me from doing them. This optimisation can only be temporary to this specific task of psuedocode, and holistically, is only applicable because I am at once the project manager and a delegate.

(Appended note 01/07/2016): comically, to look at a past project (which was discontinued), I have vigorous memory of trying to find a good stopping point, but being continually delayed. Within the source files is buried a comment: "// find a good stopping point and stop", which I indeed adhered to, leaving the projects guts wide open for an autopsy. This old source offers insight into the nature of the development environment of this project, namely, the massive distribution of internal TODO comments and asides.

In reading the comments, my thoughts that extrinsic documentation and logging can be quite damaging to actual productivity have been affirmed, since it becomes patent that many things can be managed intrinsically. The problem is that the evidence disappears iteratively: there is not a scenario where a release build would have such a massive distribution of informal comments. Wholly, this appendage serves to say that I do plenty that goes justifiably unlogged, for it is handled internally.

…

1.21. AUTOMATED MANUAL TESTING
[05/06/2016]

In the past week (or thereabouts), the project management technique of doing things rather than talking about doing things has been overwhelmingly successful. At this moment, I have finished the logic of all twenty-two modules, and have deskchecked all but four. In the interim, there hasn't been all that much exciting to log anyway: occasionally, I would have to think for a moment in order to create an algorithm. But I never thought of anything particularly exciting. Sometimes I made mistakes in algorithms, then realised the mistakes, and rectified them. This isn't very exciting. After some time, each routine was developed, and if my thoughts on each are sought, they can be found therein (`III. 4.).

Surprisingly, the testing process has hereto been more engaging and challenging than the pseudocode development (although the mechanical aside was enjoyable). Due to things such as the difficulty and length of each test run-through, some aspects of the solution have been "deskchecked" automatically. I justify this in that to assert a module as being correct, regardless of how it is done, is still to assert it as correct, and in the situation of emulation, the logic behind the tests is tangible to any happenstance observer. There is some vindication in terming these tests as being computer aided and therefore CASE tests, but this is not what is of most importance, rather, it can be said summatively that the use of JavaScript emulations anywhere during the development in areas other than the source is classifiable as an act of using software tools to develop software. Such help needn't be termed "CASE" when I create the tools myself. Indeed, without the selection and use of this software, at once the design and the development of the P.L.E solution would be much more of a struggle.

At this moment, I am still significantly behind schedule. It is no longer appropriate to neglect the creation of logs to the extent that they have been neglected over the past week, rather, it must simply be made true that I work on the project consistently.

1.22. FAILURE IN MODELLING 0/0
[06/06/2016]

In terms of progress, today, I actuated a few tests without much exciting occurring. As an aside, an exploration of indeterminacy was carried in hopes of figuring some way of explaining why 0/0 is in a superposition. I didn't arrive at anything that is explainable, and will discontinue my efforts.

…

1.23. A REWORKING OF THE PROBLEM ANALYSIS
[08/06/2016]

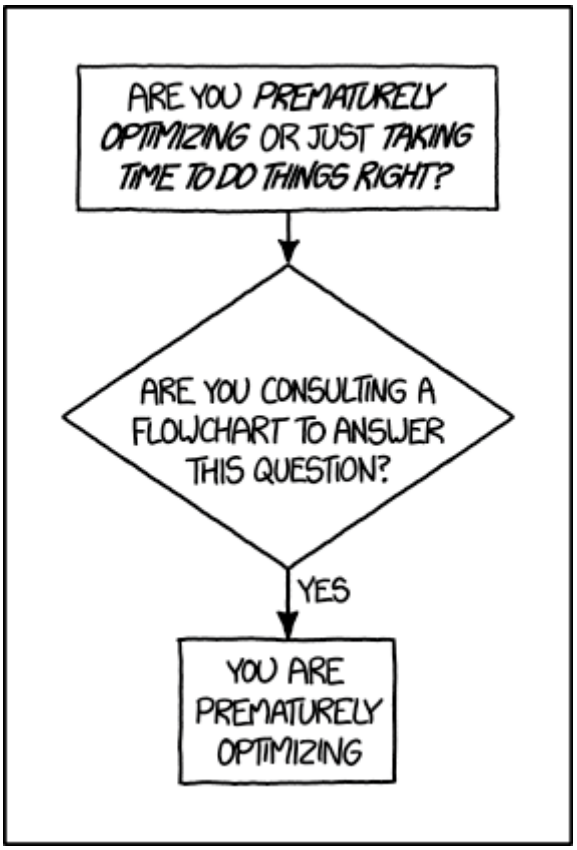
Today, I made some amendments to the problem analysis. In justification, it was in considering the nature of the problem, and was certainly in need for a rewrite both in terms of formatting and content. I spent a good deal of the day trying to prove t = E' / (E' - E), which is the premise of the project, and,

eventually, I was able to come to the same conclusion as Boole through the same means, and apply it to the problem specifically.

I then worked on applying reason (as an abstract discipline) to the call chart in the preliminary analysis, `III. 5.1., which worked surprisingly well, and I was able to make many inferences from the chart. It struck me yesterday that CASE tools would be beneficial in constructing such a chart and reaching conclusions, but, such a tool to the exact purpose was not found in post. I considered for a moment making one of my own, but eventually rose to refutation, since it would be nonsensical to create a large solution to help with the construction of a very small component of another large solution. It is left that I finish the operational analysis, considering parallelisation, space, and memory, and then, I can move on into the next stage.

1.24. OPTIMISATION
[09/06/2016]

Today, I read an xkcd comic at http://xkcd.com/1691/, which eerily represents today's work on parallelisation. I am, indeed, spending too much time on planning, and ought to hurry myself up.



1.25. PARALLEL RATIOS
[10/06/2016]

A JavaScript tool, which, essentially is a homemade CASE tool, was deemed appropriate for the generation and analysis of results from parallelisation. The following was created to aid in the generation of results:

```
function getSpeedup(α, β, γ, δ, ψ, n)
{
    α = eval(α);
    β = eval(α);
    γ = eval(α);
    δ = eval(α);
    speedup = (α*β + γ + δ)/(α*β + γ + δ - (β + γ)/ψ);
    return speedup;
}
```

But after some time of experimentation and testing, I found an obvious bug: $\alpha = \beta = \gamma = \delta$. This was rectified quite simply to create the following, arriving at the results in `III. 5.2.:

```
function getSpeedup(α, β, γ, δ, ψ, n)
{
    α = eval(α);
    β = eval(β);
    γ = eval(γ);
    δ = eval(δ);
    speedup = (α*β + γ + δ)/(α*β + γ + δ - (β + γ)/ψ);
    return speedup;
}
```

But, further experimentation yielded the proof, somehow, incorrect (which is poor form):

The complexity, Θ, of each module 12 and 17, with respect to n, is given by:

$\Theta_{12} = (n + 1)2^n + n (+ C);$ (1)

$\Theta_{17} = 2 \cdot 2^n + 2n^2 + 2n (+ C);$ (2)

From (1) and (2), ignoring the incalculable constant overhead C:

$\Theta_{\Sigma} = \Theta_{12} + \Theta_{17};$

$\Theta_{\Sigma} = (n + 1)2^n + 2 \cdot 2^n + 2n^2 + 3n;$ (3)

Considering that there is really only worth and ease in parallelising terms containing 2^n, a generally applicable formula can be created :

$\Theta_{\Sigma} = \alpha(n)\beta(n) + \gamma(n) + \delta(n);$ (4)

Let it be said that $\beta(n)$ and $\gamma(n)$ are parallelisable, whereas $\alpha(n)$ and $\delta(n)$ are not, then, (4) can be reconstructed introducing distributed processes, and a number of processors, ψ,

$\Theta_{\Sigma} = \psi \cdot \alpha(n)\theta_{\beta}(n, \psi) + \psi \cdot \theta_{\gamma}(n, \psi) + \delta(n);$ (5)

It should be noted that both functions θ are non-concurrent, that is, one cannot begin to be executed until the other has been completed, thus, each has full processor dedication.

Knowing (5) = (4),

$\psi \cdot \alpha(n)\theta_{\beta}(n, \psi) + \psi \cdot \theta_{\gamma}(n, \psi) + \delta(n) = \alpha(n)\beta(n) + \gamma(n) + \delta(n);$

$\psi \cdot \alpha(n)\theta_{\beta}(n, \psi) + \psi \cdot \theta_{\gamma}(n, \psi) = \alpha(n)\beta(n) + \gamma(n);$

$\alpha(n)\theta_{\beta}(n, \psi) + \theta_{\gamma}(n, \psi) = \psi^{-1}[\alpha(n)\beta(n) + \gamma(n)];$ (6)

But, (6) is clearly satisfied by the two symmetrical substitutions,

$\theta_{\beta}(n, \psi) = \beta(n)\psi^{-1};$ (7)

$\theta_{\gamma}(n, \psi) = \gamma(n)\psi^{-1};$ (8)

Adding (7) and (8),

$\theta_{\Sigma} = [\beta(n) + \gamma(n)]\psi^{-1};$ (9)

The fundamental equation of parallelisation is that speedup, S --

$S = \Theta_{\Sigma}(\Theta_{\Sigma} - \theta_{\Sigma})^{-1};$ (10)

Applying (10) across (9) and (4):

$S = [\alpha(n)\beta(n) + \gamma(n) + \delta(n)][\alpha(n)\beta(n) + \gamma(n) + \delta(n) - \{\beta(n) + \gamma(n)\}\psi^{-1}]^{-1};$ (11)

Methinks, somewhere, I made the assumption that ψ is at once a percentage, and also a number of processors, or that S should be its own reciprocal, &c:

```
function getSpeedup(α, β, γ, δ, ψ, n)
{
    α = eval(α);
    β = eval(β);
    γ = eval(γ);
    δ = eval(δ);
    speedup = 1/((α*β + γ + δ - (β + γ))/(α*β + γ + δ - (β + γ)/ψ));
    return speedup;
}
```

But regardless, things aren't working very well, and although being quite tempted to correct this, especially since the formula can be reasoned through fairly thoroughly asymptotically, and using other neat interpolative techniques;-- I am still well behind schedule, as conferred by the magnificently useful Gantt chart. Instead, more rudimentary figures must be used to perform a basic analysis.

1.26. PAST MISTAKES IN IMPLEMENTATION [11/06/2016]

In the prior work, Egg Classification Report for NSW Egg Board, 19th February 2016, I made a fundamental design mistake, centred on an incorrect implementation of IOStream. IOStream, really, should only be needed in the testing of the solution holistically, not at lower levels. I am paying careful attention to this during implementation. Essentially, and verily, this is the only summary that can be

afforded, I had in the past made the mistake of returning results of a main method call to a spoofIOstream via a parameter unnecessarily, which led to there having to be some careful treading throughout implementation and documentation. Thus, during the first stage of implementation (these stages are things which have been mapped out in my head, and due to their simplicity, needn't be planned), there will be no IOStream.

Visual Studio was chosen for the development environment, and the selection of Python of a language was today formalised -- Python's IDLE environment is terrible. Verily, without Visual Studio, the task of concurrent work on multiple files would be impractical. Visual Studio, however, offers a vastly redundant (to my purposes in this development) amount of features, most of which I have no idea what they do, or how to use them. Python is a terribly ugly language without a good development environment. I contacted the client regarding language choice, and he supported Python.

1.27. VISUAL STUDIO
[12/06/2016]

The possible state / coefficient map / simplified expression table in `IV. 2.1. is exponentially exponential in the amount of rows. With only one additional symbol, there would need to be 256 rows. This yields that each system containing, say, 5 symbols, can be in any one of 4,294,967,296 configurations, so perhaps dynamic programming would have been a ridiculous idea, and I am very glad not to have wasted time in explicitly analysing feasibility. This would however make an interesting problem of multidimensional symmetry.

I encountered the error in `V. 2.1.. Python clearly isn't a fan of folders. On top of this, PTVS (Python Tools for Visual Studio), in its current version, assumes that what ought to be correct is in fact correct. This means that IntelliSense is:

```
Unable to resolve "Relation.Relation". IntelliSense may be missing for this
module.
```

A quick Google search of this error, https://pytools.codeplex.com/discussions/543107, *Python Tools for Visual Studio - IntelliSense isn't aware of my Django apps in PTVS 2.1*, accessed on this day, tells how this is an issue of PTVS 2.1, and has since been rectified in version 2.2. Although I already had version 2.2 installed, a newer version, 2.2.2, was available, so I installed that, but the error persisted. What then? I tried updating Python, since Python version control isn't already hard enough … (but Visual Studio should actually make version control possible). Same issue, but now I will be doing the coding in Python 3.5 as opposed to 3.4 -- but wait, the error has changed:

```
Unable to resolve "Relation.Relation".
The completion DB for the active environment needs to be refreshed,
and IntelliSense may be missing until that has completed.
See the Python Environments window for details.
```

Then Microsoft.PythonTools.Analyzer.exe appeared to have a memory leak, and I restarted the computer … But still, reopening the project in Visual Studio saw the process hang at a fairly constant 20% CPU usage and 1 GB of RAM. Ending this task individually allows Visual Studio to be used independently, but, after investigation, it was turned up that this process is a result of PTVS trying to update the Python environment. In short, the process was able to finish before my PC overheated, and needn't be executed again, as by https://github.com/Microsoft/PTVS/issues/1016, *Python Tools Background Analyzer Automatically Starts Up · Issue #1016 · Microsoft/PTVS · GitHub*. But, after updating the definitions, still,

```
Unable to resolve "Relation.Relation". IntelliSense may be missing for this
module.
```

Some time later, after I had accepted the problem as being unsolvable, I found an internal updating tool within Visual Studio. I had some interest in how Visual Studio updates worked (in its infinite extensibility), so I investigated -- to see, instantly, that PTVS 2.2.3 is the newest version, not 2.2.2. -- 7 GB of updates later … Still no. Instead, I am just using a spoof "import Relation" command, which doesn't actually do anything, but tricks IntelliSense into thinking it does, and therefore, I can read helpful suggestions that are almost worth the whole ordeal.

1.28. DOCSTRINGS AND DEBUGGING [13/06/2016]

An opening snippet of the Relation class looks like:

```
class Relation:
    """\
Stores and manipulates a connection of Boolean symbols.
Unsafe unless Relation::isValid(this) returns true.
    """
    def __init__(this, constructor):
        this.data = constructor;
```

The docstring, beginning with the three quotation marks, must necessarily be left-aligned, otherwise, Python parses it as though the whitespace is part of the string. Docstrings are a form of internal (but not intrinsic) code documentation: they are like comments, except they look terrible, but provide metadata for the program in equipoise. Were I just to be using comments, everything would look a lot better, except the metadata is lost:

```
class Relation:
    # Stores and manipulates a connection of Boolean symbols.
    # Unsafe unless Relation::isValid(this) returns true.
    def __init__(this, constructor):
        this.data = constructor;
```

And even more so, I think that the comment should be migrated above the class / function it is commenting on, as next:

```
# Stores and manipulates a connection of Boolean symbols.
# Unsafe unless Relation::isValid(this) returns true.
class Relation:
    def __init__(this, constructor):
        this.data = constructor;
```

If docstring functionality is truly sought, it can be added in post fairly easily, but otherwise, I value consistent indentation over further IntelliSense information while hovering. The program will be externally documented to the end that a docstring would otherwise purpose.

Considering now the debugging I did today (clearly not all debugging will be discussed or mentioned at all, just chosen examples), it is detailed in `V. 2.2. how I forgot to increment / decrement a counter in a loop. This is a day one programming mistake, but it is easily justifiable. Throughout all of the pseudocode in this report (by my memory), I do not once use a *while* loop. I am not accustomed to using them. *For* loops feel more applicable to me in most scenarios, so their use is wont. I write a lot in C++, so let this language be used for demonstration: the following two snippets are functionally equivalent,

```
int numSandwiches = 8;
Sandwich *sandwiches = new Sandwich[numSandwiches];
populateSandwiches(sandwiches);
for (int sandwichNum = 0; sandwichNum < numSandwiches; ++sandwichNum)
{
    makeSandwich(sandwiches[sandwichNum]);
}
delete[] sandwiches;
```

```
int numSandwiches = 8;
Sandwich *sandwiches = new Sandwich[numSandwiches];
populateSandwiches(sandwiches);
int sandwichNum = 0;
while (sandwichNum < numSandwiches)
{
    makeSandwich(sandwiches[sandwichNum]);
    ++sandwichNum;
}
delete[] sandwiches;
```

Methinks the first is more sensible, but Python -- in its inexorable mediocrity -- does not support *for* loops (one must not get confused between *for* loops and for-in range repetition). My habit was therefore broken, and so, I erred -- not from lack of knowledge or experience, but by old habits.

Before I realised this invoked mistake, I was in the process of writing debugging output statements to help locate the error. This is exemplification of a method of systematic error location (and thus removal), and isn't especially exciting, but must be expounded so as to meet the primary purpose of this work. Visual Studio, and its honestly amazing tools, offers several means through which

debugging can be performed, and, surely, these methods will be used to detect other bugs that escape the mention of this report (to write about everything would kill all productivity in coding).

Opening the context menu (right clicking) on source in the PTVS environment gives some hasty options for debugging: insert breakpoint, insert tracepoint, and run to cursor. Adding a breakpoint yields further options such as to break on conditions, and adding tracepoint gives options to print contents of variables to the screen during execution. In debugging mode (which is hastily activated by pressing F5), most importantly, the contents of variables and members can be examined, as well as the call stack viewed, which offers a great deal of insight into what a program is actually doing while it is running, thus exposing errors.

Progress-wise, I finished QuantumCounter.py and testQuantumCounter satisfactorily, so as to allow the continued development of Relation.py.

Also, the addition of a __init__.py file as the startup file in the PLE directory, containing simply the following, seems to solve a lot of module inheritance problems (but not IntelliSense),

```
# Nicholas Killeen

import testPLE;

def main():
    testPLE.testPLE();
main();
```

Another error which occurred during development (which I touch on briefly) is in the Relation.__eq__ function, and was systematically found through the use of debugging output statements:

```
# iterate through every possible state of coefficients, if pairs
# do not match from both relations, they are not equal
numCoefficients = 2 ** numSymbols;
coefficientNumber = 0;
hasEncounteredDiscrepancy = False;
while (coefficientNumber < numCoefficients):
    relationA = Relation();
    relationA.copyFrom(this);
    relationA.insertArguments(coefficientMap);
    relationB = Relation();
    relationB.copyFrom(arg);
    relationB.insertArguments(coefficientMap);
    if (relationA.evaluate() != relationB.evaluate):
        hasEncounteredDiscrepancy = True;
    coefficientMap.increment();
    coefficientNumber += 1;
    return not hasEncounteredDiscrepancy;
```

Closer analysis of the test log reveals it to have been quite useful (highlighted):

```
Testing Relation ... Traceback (most recent call last):
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\__init__.py", line 7, in <module>
    main();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\__init__.py", line 6, in main
    testPLE.testPLE();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\testPLE.py", line 19, in testPLE
    testRelation.testRelation();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\Relation\testRelation.py", line 125, in testRelation
    assert((relation15 == relation16) == True);
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\Relation\Relation.py", line 46, in __eq__
    if (relationA.evaluate() != relationB.evaluate):
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\Relation\Relation.py", line 72, in evaluate
    return eval(evaluableString);
  File "<string>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'int' and 'list'
Press any key to continue . . .
```

Surprisingly, the line numbers also match. I seem to have memory of them not in the past because of how Python dismisses comments as not being enumerable lines, but I have been unable to reproduce this. --

Yet,

Rectification of the obvious error still yields the exact same error message,

```
TypeError: unsupported operand type(s) for -: 'int' and 'list'
Press any key to continue . . .
```

So it becomes obvious that the error is in the evaluate function -- but wait, I have already tested that to some extent:

```
relation7 = Relation.Relation("1");
assert(relation7.evaluate() == 1);

relation8 = Relation.Relation("0");
assert(relation8.evaluate() == 0);

relation9 = Relation.Relation("(0)(1)");
assert(relation9.evaluate() == 0);

relation10 = Relation.Relation("(0)(1) + 1 + (0)(1 - (1))");
assert(relation10.evaluate() == 1);

relation11 = Relation.Relation("(1)(1)(0) + (1)(1 - (1))(1 - ((0)(1"
    "))) + (1 - (1))(1 - (1))");
assert(relation11.evaluate() == 0);
```

```
relation12 = Relation.Relation("(0)(1 - (0)) + (0)(1 - (1)) + (1)("
    "1 - (0))(1 - (1 - (1)))");
assert(relation12.evaluate() == 1);
```

By simple inference, peradventure, it must be that the fault lies in the format of the relation, not the evaluate function. Since I have not created the isValid function, the Relation must manually be examined (by placing a breakpoint on the highlighted line and entering debug mode):

```
relationA  <Relation.Relation.Relation object at 0x00000269BBB40E80>  Relation
    data   '(1)(1 - [1]) + [1](1 - (1))'                              str
relationB  <Relation.Relation.Relation object at 0x00000269BBB40F28>  Relation
    data   '(1 - [1])'                                                str
```

This verifies the hypothesised correctness of the evaluate method. Further investigation (or by haphazard instinct) turns up:

```
coefficientMap <QuantumCounter.QuantumCounter.QuantumCounter object at
    0x0000013A86D40EF0>   QuantumCounter
    bitStates    [-1]     list
        [0]      -1       int
```

Which makes any sceptical person think: wait -- there is only one bitState, and it is in the quantum position, while in both of the Relations, there are two symbols, even though one is implicit…

A sigh of realisation is warranted.

To the question of causality, I answer that we are not keeping track of how many symbols there actually are, merely, assigning the length of the symbolList to the numSymbols variable. I offer as evidence the locals list:

```
symbolList        [1]   list
    [0]           1     int
numSymbols        1     int
coefficientArray  ['']  list
    [0]           ''    str
```

And now I cite the true erroneous line:

```
coefficientArray = [""] * numSymbols;
```

Which should read:

```
coefficientArray = [""] * (max(symbolList) + 1);
```

Yet, …

At the same breakpoint, coefficientMap, despite now correctly holding two elements, assumes both of the bits are in quantum superpositions [-1, -1]. Path analysis reveals that within some prior conditional, the highlighted line don't get executed when they ought to:

```
while (symbol < numSymbols):
    if (symbolList.count(symbol) == 1):
        coefficientArray[symbol] = 0;
    symbol += 1;
coefficientMap = QuantumCounter.QuantumCounter(coefficientArray);
```

It becomes immediately apparent that numSymbols is again to blame, and a fix can be fully implemented by reverting the old change, and changing the assignment of numSymbols as is next demonstrated:

```
numSymbols = 0;
if (len(symbolList) > 0):
    numSymbols = (max(symbolList) + 1);
```

Then suddenly,

```
Testing Relation ... passed!
```

Some other undocumented changes had to be made to avoid exponentiality proportional to the largest symbol, as opposed to the number of symbols, since, printing the contents of coefficientMap causes clear repeats (highlighted):

```
[-1, 0, 0, -1, -1, -1, 0, -1, -1, -1, -1, 0]
[-1, 0, 0, -1, -1, -1, 0, -1, -1, -1, -1, 1]
[-1, 0, 0, -1, -1, -1, 1, -1, -1, -1, -1, 0]
[-1, 0, 0, -1, -1, -1, 1, -1, -1, -1, -1, 1]
[-1, 0, 1, -1, -1, -1, 0, -1, -1, -1, -1, 0]
[-1, 0, 1, -1, -1, -1, 0, -1, -1, -1, -1, 1]
[-1, 0, 1, -1, -1, -1, 1, -1, -1, -1, -1, 0]
[-1, 0, 1, -1, -1, -1, 1, -1, -1, -1, -1, 1]
[-1, 1, 0, -1, -1, -1, 0, -1, -1, -1, -1, 0]
[-1, 1, 0, -1, -1, -1, 0, -1, -1, -1, -1, 1]
[-1, 1, 0, -1, -1, -1, 1, -1, -1, -1, -1, 0]
[-1, 1, 0, -1, -1, -1, 1, -1, -1, -1, -1, 1]
[-1, 1, 1, -1, -1, -1, 0, -1, -1, -1, -1, 0]
[-1, 1, 1, -1, -1, -1, 0, -1, -1, -1, -1, 1]
[-1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, 0]
[-1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, 1]
[-1, 0, 0, -1, -1, -1, 0, -1, -1, -1, -1, 0]
...
```

I extend today's log to express how wonderful it was to read the following error message.

```
Testing FactDatabase ... passed!
Testing QuantumCounter ... passed!
Testing Relation ... Traceback (most recent call last):
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\__init__.py", line 7, in <module>
    main();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\__init__.py", line 6, in main
```

```
    testPLE.testPLE();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\testPLE.py", line 19, in testPLE
    testRelation.testRelation();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\Relation\testRelation.py", line 199, in testRelation
    asert(relation29 == relation30);
NameError: name 'asert' is not defined
Press any key to continue . . .
```

And on fixing (unit tests hadn't been running successfully for some time):

```
Testing Relation ... passed!
```

1.29. ON THE CONNECTION OF BOOLEAN ALGEBRA AND ABSTRACTION [14/06/2016]

Firstly, a simple error log (just to demonstrate that I am still doing things):

```
    relation35 = Relation.Relation("[0] + [1](1 - [0])");
    # ...
    relation38 = Relation.Relation("(0) + ((([1])))");
    relation39 = Relation.Relation("(1 - [0])");
    # ...
    assert(relation35 * relation38 == relation39);
```

```
Testing Relation ... Traceback (most recent call last):
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\__init__.py", line 7, in <module>
    main();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\__init__.py", line 6, in main
    testPLE.testPLE();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\testPLE.py", line 19, in testPLE
    testRelation.testRelation();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\Relation\testRelation.py", line 221, in testRelation
    assert(relation35 * relation38 == relation39);
AssertionError
Press any key to continue . . .
```

This is very surprising behaviour, so playing with the interpreter reveals:

```
>>> relationA = Relation("[0] + [1](1 - [0])");
>>> relationB = Relation("(0) + ((([1])))");
>>> relationActual = relationA * relationB;
>>> relationActual.data
'([0] + [1](1 - [0]))((0) + ((([1]))))'
>>>
```

So the condition is failing because:

```
Relation("([0] + [1](1 - [0]))((0) + ((([1])))))");
```

allegedly, doesn't equal

```
Relation("(1 - [0])");
```

This can be checked on pen and paper, quite easily:

([0] + [1](1 - [0]))((0) + ((([1])))) = (1 - [0]);

LHS = ([0] + [1](1 - [0]))([1]);
[0][1] + [1][1](1 - [0]);
[0][1] + [1](1 - [0]);
[1];
Which is clearly not workable to RHS, ∴, the two Relations are not equal.

This verification means I was wrong, but the program I wrote was right (rather not necessarily incorrect). The test case has since been changed.

Another brief aside, I do not feel very comfortable with this snippet of code:

```
# Evaluates a Relation that contains no symbols.
# Unsafe method; must not be called unless:
#  - Relation::isValid(this) -> True.
#  - Relation::getSymbolList(this) -> [].
def evaluate(this):
    evaluableString = this.data.replace(")(", ")*(");
    return eval(evaluableString);
```

There is no input validation here whatsoever, and in fact, for the Relation class wholly, all input must be validated externally (which is imperative to the performance of the program). This would not be acceptable if the Relation functions are directly callable by the user, but, this is not the case -- these functions belong to the core, not the external interface. Regardless, there is a risk if this module is reused in other programs, so it must be clearly stated in comments that the methods are potentially unsafe (by which is meant they do not necessarily handle their own errors, or can be used to produce unexpected behaviour). Also, on first instinct, it feels like I should create a function isEvaluable to return true if getSymbolList() → [].This thought is rebuked by the knowledge that the function would never actually be called for the purposes in the PLE. It will be added if found necessary for the CLI.

I have now practically finished the Relation class, and with such completion, my thoughts have been directed to what is more so the philosophy of Boolean algebra (but time compels me to be pithy and unrefined in expression). Without citation, I say that people, or people in general, think Boolean algebra is dualistic. Such viewpoints held by people in my potentially unfounded projections are wholly unfounded -- although Boole did not succeed (by my memory, and I don't know of anyone

who has done it better) in establishing the exact meaning of the indeterminate state 0/0, his system does support abstraction and unknowns in this form to he only possible end. This intrinsic behaviour is manifested in the Relation class: within the class, there is no mention of indeterminism (other than the coefficientMaps following the QuantumCounter data structure, which could have easily been represented in many other forms), yet, the class is still able to have placed atop it indeterminate states. This is all very interesting, but not relevant to the forwarding of the project, and so I cease continuity.

I would like now to jot down a paragraph on "accomplishments of today", but to keep track would be a waste of time. To log "I wrote lines 10 - 20 in XXX.py, and tests on lines 100-140 in YYY.py", severely impacts my sanity, and the progression of the project. Being at once the project manager and the sole worker, the necessity of logging tasks is not so harshly enforced, though, Visual Studio does allow me to track my progress through computerised project management features (which it is beyond purpose to log).Visual Studio highlights lines and files that have been modified in an instance of a project, which is a surprisingly helpful visualisation for how much work has been performed, and in what areas. There is, howbeit, no way of representing this textually, so it is left out of the logbook.

1.30. REGULAR EXPRESSIONS
[15/06/2016]

Using the "re" Python module, regular expressions can be implemented to greatly reduce the amount of lines necessary in the validating method Relation::isValid(this), but, it is not understandable that a reader can make any sense of this:

```
/[^0-9 \[\]()+-]/
```

Myself, I don't know regular expressions, because it is painfully pithy to express and learn (although beneficial as soon manifestly). Since the isValid function is only called in overhead, it is allowed to be fairly expensive, so, the functionality of regular expressions can be expanded over a number of lines without detrimental effects, and also, with success in embellishing the solution in terms of readability, which is tacit in the problem definition. Because I don't know regexp, I first perceived that some oddities have to be mechanically generated, as below:

```
VALID_SYMBOL_COMBINATIONS = ["+(", "+[", ")+", "]+", "-(", "-[",
    ")-", "]-", ")(", ")[", "](", "]["];
```

I began to struggle expressing the rules or Relation syntax, as obvious by the mess of a first draft (which does not constitute a valid working environment):

```
def isValid(this):

    DIGITS = "0123456789";
    VALID_CHARACTERS = "0123456789 +-()[]";

    dataString = this.data;
```

```
    dataStringNoSpace = dataString.replace(' ', "");
    isValid = True;
    if (len(dataStringNoSpace) == 0):
        isValid = False;

    currentDepth = 0;
    hasUnresolvedSquareBracket = False;
    for character in dataString:
        if (VALID_CHARACTERS.count(character) == 0):
            # if any character is not found in the valid list
            isValid = False;

        if (character == '['):
            if (hasUnresolvedSquareBracket):
                # '[' characters cannot be nested
                isValid = False;
            hasUnresolvedSquareBracket = True;

        if (character == ']'):
            hasUnresolvedSquareBracket = False;

        if (hasUnresolvedSquareBracket):
            if (DIGITS.count(character) != 0):
                isValid = False;

        #
        if

        ?
    # .

    if (count != 0):
        pass;
        pass;
        pass;

    # valid expressions must also exist within the Boolean domain,
    # {0, 1}, having the condition n = n^2
    if (isValid):
        if (this != this * this):
            isValid = False;
    return isValid;
```

Much of the issue is because Python, unforgivably, cannot parse leading zeros:

```
>>> 01 + 1
  File "<stdin>", line 1
    01 + 1
     ^
SyntaxError: invalid token
>>>
```

I then began to write, echoing the structure of eval.cpp --

```
# Class to model Relation::isValid(this).
class _RelationTraverser:
    def __init__(this, data):
        this.data = data;

    def _getMaxDepth(this):
        maxDepth = 0;
        depth = 0;
        for character in data:
            if (character == '('):
                depth += 1;
            elif (character == ')'):
                depth -= 1;
            if (depth > maxDepth):
                maxDepth = depth;
        return maxDepth;

    # Returns the start and end index of the innermost set of brackets.
    def _getInnerBracketParameters(this):
        maxDepth = _getMaxDepth();
        dataLength = len(this.data);
        depth = 0;
        index = 0;
        beginIndex = 0; # default
        endIndex = dataLength - 1; # default
        indexWithMaxDepth = 0;
        while (index < dataLength):
            if (character == '('):
                depth += 1;
                if (depth == maxDepth):
                    beginIndex = index;
            elif (character == ')'):
                if (depth == maxDepth):
                    endIndex = index;
                depth -= 1;
        return {"beginIndex": beginIndex, "endIndex": endIndex};

    def _collapseInnerBracket(this):
        innerBracketParameters = _getInnerBracketParameters();
        beginIndex = innerBracketParameters["beginIndex"];
        endIndex = innerBracketParameters["endIndex"];


    def isTraversable(this):
        return True;
```

But realised that this may be unnecessarily complicated. The complexity of this problem -- which appears to be such a trivial matter -- is simply because Python treats '0' so arbitrarily different from other digits, and due also to the existence of whitespace with meaning varying with respect to context.

Another haphazard attempt saw the snippet below being generated (where my goal is obvious, although the source is well incomplete):

```
if (isValid):
    acceptedNextSymbols = VALID_CHARACTERS;
    lastEncounteredBracket = "";
    for character in dataString:
        if (acceptedNextSymbols.count(character) != 0):
            isValid = False;

        if ("0123456789".count(character) != 0):
            acceptedNextSymbols = "0123456789 +-";
        elif (character == 0):

        elif (character)
```

This final failed attempt provided merit for pseudo-proper planning, and an approximate definition of steps:

1. Recursively amalgamate all digits, such that "120 + 100" becomes "n + n"
2. Recursively reduce all meaningless whitespace, so "(1) (2)" becomes "(1) (2)"
3. Recursively replace all symbols, "[n]", with "(n)"
4. Recursively replace all "(n)(n)" with "(n)"
5. Recursively replace all "(n)" with "n"
6. Recursively replace all "n+n" and "n-n" with "X"
7. Remove all "X" characters,

In the scenario where the string is empty, it is indicated as true that the constructor is valid. Clearly, this process is relying on a lot of "recursive replacement / reduction", so, this ought to be implemented as a subroutine. It can further be noticed that this tokenisation process is analogous to the translation process of higher level source code to machine code, where syntactical analysis is performed, and then active lexical analysis reduces the string.

1.31. "# IS THIS WHERE I SHOULD DO E' / (E' - E)" [16/06/2016]

The productive half of the day is over, and at least mostly, I have managed to finish the whole core, do some mild refactoring for readability (thanking the unit tests for the confidence they instil as I begin to rip my past code apart), create rigorous tests, and document a few things. I am worried to explicitly document the intrinsic workings of the system, say, in a data dictionary (at this point), because there is great potential for modification, and maintenance work is much harder to document than changing an unpublished document.

I would call the first stage of the solution complete were it not for one line, appended at the end of FactDatabase.py, exactly reading:

```
# is this where i should do E' / (E' - E) ################################
##################################################
```

I ask the reader, firstly, not to judge the capitalisation of 'i', and I answer that I known not whether that is how I perceive the capitalisation in that circumstance as being correct. But it certainly looks strange. Getting to the root of the issue though, not just bikeshedding, my planning has failed to prepare me in positioning the figuratively main method of the program. Without the existence of this method, the source code is merely a toolkit which can be conveniently used to create the solution. But, putting the method in the core causes severe problems when it comes to parallelisation, and how the two user interfaces behave. I will now express my argument.

It is truly patent that the responsibility of solution should belong wholly to the caller, since it is the caller who controls the way in which the process is performed. The caller says "go, now stop, resume, and give up -- now do half of the work". To reflect this functionality at such a low level would be an absurd misplacement of responsibility. The same overhead would need to be performed on each parallel call to the method, since the expensive local variables will go out of scope. This method makes control of process very difficult, and should be avoided thusly.

Perhaps, in the near future, some more interfacing functions can be added to make things neater, but I think I am ready to move onto the CLI implementation.

1.32. INFORMAL PLANNING
[17/06/2016]

Much of today was simply spent thinking about how the Interface is going to be implemented (it should be noted that much of the functionality will reflect what is planned in the pseudocode). I fiddled with the file structure so that it is just right. In the Gantt chart, I still logged today as a full day, and methinks that is justifiable, but, I do not myself have mind to justify it -- on paper, I scribbled down a lot of nonsense so that my mental picture of the system could be reformed. None of this is documentable in the actual solution, but ought nonetheless to be reproducible as evidence for working on the system.

1.33. INTERFACING
[18/06/2016]

The interface is going along well -- I have added two new files in a folder Interface, following convention, being called Interface.py and testInterface.py, and likewise for "Processor". Things are starting to fit together.

To arbitrarily log some debugging, I wrote a round of unit tests, which caused the program to terminate. The error was quickly traced to the highlighted line(s):

```
relation5 = Relation.Relation("[1]");
interface1.setRequest(relation5);
request = interface1.getRequest();
assert(request == relation5);
```

```
assert(len(errorList) == 5);
```

And

```
def getRequest(this):
    if (this.request == None):
        this.errorOutputMethod("Failed to get request.");
    return this.request;
```

The expressive part of the error message reads (since I have grown mind to shorten them):

```
AttributeError: 'NoneType' object has no attribute 'getSymbolList'
```

And it becomes known that the program is trying to compare a Relation with a NoneType through its Relation-Relation comparison method. The fix for this isn't easily expressible, because it involves changes in a few places, but, in brevity, the getRequest method is now protected with a flag that is externally controlled:

```
def getRequest(this):
    request = None;
    if (this.isRequestValid):
        request = this.request;
    else:
        this.errorOutputMethod("Failed to get request.");
    return request;
```

There is also a lot of white box prodding going on, which wasn't necessarily all too present in the earlier stage of development. This is because much of the routines that have to be checked produce side effects, or are only intermediaries in a much larger process, whose results are therefore not verifiable unless the whole process is actuated. The nature of this, really, isn't conducive to easy logging, and to dump code and explain it would be to extend the length of this report unnecessarily. In best summary, it involves examining the intrinsic properties of classes under certain conditions. This involves the printing of a lot of variables to the screen, or examining the contents of variables using VS tools.

1.34. CONFERENCE
[19/06/2016]

I asked the client, my brother, what degree of functionality must be present in the CLI as opposed to the GUI. He said that he cannot imagine scenarios whereby a CLI would be preferred over a GUI, and whence, I shall make the CLI fairly rudimentary -- which makes design a lot easier.

To develop a GUI, I will be using the PySide Python package -- but wait:

```
The following error was encountered: TF400813: Resource not available for
anonymous access. Client authentication required.Installing 'pyside'
Collecting pyside
  Downloading PySide-1.2.4.tar.gz (9.3MB)
```

```
    Complete output from command python setup.py egg_info:
    only these python versions are supported: [(2, 6), (2, 7), (3, 2), (3,
3), (3, 4)]

    ----------------------------------------
Command "python setup.py egg_info" failed with error code 1 in
C:\Users\Nicholas\AppData\Local\Temp\pip-build-l7akt1vz\pyside
You are using pip version 7.1.2, however version 8.1.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip'
command.
'pyside' failed to install. Exit code: 1
```

Development will thus continue in Python 3.4 (In my experience, Python has the ugliest version control of all the things).

I was just introduced to the Visual Studio HTML / CSS / JS editor, and it is incomprehensibly pretty. One formatting error has been encountered, namely, when opening a HTML file for reading in Python, there is a small string of garbage to begin the stream:

```
'ï»¿<!DOCTYPE html>\n\n<html lang="en">\n<head> ...
```

The problem was easily solved:

```
htmlContent = htmlContent[htmlContent.find("<!DOCTYPE html>"):];
```

Really, I have left the rest of the program in a shoddy state while I work on the GUI. This is really as a result of a poor Repository-Processor-Interface model. Things may be refactored heavily later, but for now, things are being left as they are -- I am planning to redefine the Interface as I actually need it in the GUI. I am thinking, actually, of removing parallel processing altogether -- but this would be a clear breach of the spec: (iv) The main process of the P.L.E must be manually parallelisable. For ease -- let it be changed, and justification given in the operational analysis. The client, whose input is really all that matters, agrees that parallelisation isn't necessary, provided that performance is not significantly tarnished.

Thus, parallel processing has been (is being) removed -- with use of Visual Studio tools (Visual Studio still surprises me with its resplendent aspect) such as "find all references", and "remove imports". Refactoring to the point of unit test passing took about five minutes, and then another half hour to fix some things up (and add new unit tests). Progress was slowed by the addition of the test:

```
interface3 = Interface.Interface(errorList.append);
interface3.addFact(Relation.Relation("[1](1 - [0])"));
interface3.addFact(Relation.Relation("[0](1 - [1])"));
interface2.setRequest(Relation.Relation("[1]"));
quaesitum = interface3.retrieveResponse();
assert(len(errorList) == 1);
handler = interface2.generateProcessor();
quaesitum = interface3.retrieveResponse();
assert(quaesitum == None);
```

```
assert(len(errorList) == 2);
handler();
quaesitum = interface3.retrieveResponse();
assert(quaesitum != None);
assert(len(errorList) == 2);
```

But then, I realised the mistake (after the larger part of half an hour) -- the code makes partial references to interface2 instead of interface3.

1.35. CSS
[20/06/2016]

I am getting to know (or am rather being reacquainted with the fact that) CSS is one of the hackiest languages in any sort of common use. This issue is compounded in that the PySide module used to simulate a webkit browser doesn't actually have full functionality, and is missing what I perceive to be essential features such as { display: flex; }. This means that perplexing hacks have to be made quite regularly -- but CSS documentation isn't really necessary. You can't really design the implementation of a webpage stylesheet unless you know *exactly* what you are doing -- I don't -- and it is not reasonable to think that I should, or to think that anyone partaking in any reasonably large non-web-specific project should. To model the CSS / HTML design a bit though, Mozilla Firefox is being used as a CASE tool to visualise the P.L.E skin -- namely, it is the 3d view tool that is in use.

There are really two design options I have for the GUI (in terms of functionality): the data can be represented physically as HTML attributes and nodes, or, the HTML contents can mirror data stored locally in JavaScript. Methinks I will go with the second, since it makes saving easier (perhaps with thanks to JSON), and node traversal in HTML is something I don't have much experience in.

I also gave my brother a basic run-through of the developer CLI build, demonstrating the following exchange with verbal explanation, just to exemplify that things are coming along and that the main functionality is there, it just needs to be skinned:

```
O: 1. Add a fact
O: 2. Show a list of facts
O: 3. Request a solution.
I: 1
O: Enter the fact:
I: [1](1 - [2][3])
O: 1. Add a fact
O: 2. Show a list of facts
O: 3. Request a solution.
I: 1
O: Enter the fact:
I: [2][3](1 - [1])
O: 1. Add a fact
O: 2. Show a list of facts
O: 3. Request a solution.
I: 3
O: Enter the request:
```

```
I: [1]
O: 1[2][3]
O: 1. Add a fact
O: 2. Show a list of facts
O: 3. Request a solution.
I: 3
O: Enter the request:
I: [2][3]
O: 1[1]
...
```

Otherwise, I am having difficulty styling the HTML page, generating long lines such as:

```
document.getElementById("factAppendPointer").insertAdjacentElement("beforeb
egin", relationElement);
```

This -- especially when coupled with at least three levels of indentation -- does not conform to the self-imposed 72 character limit. But, the purpose of this old limit (which is adhered to throughout the Python sources) is partially to make things obviously readable in the report, since Python is a language which does not ignore white space. HTML, CSS, and JS, conversely, all mostly ignore whitespace, so if one line of functionality overflows marginally, it may still be readable in the report without additional styling. I will have to consider this issue in post.

1.36. NA_N
[21/06/2016]

Firstly, I should probably introduce the all-caps heading notation for this report: underscores are initially illegal, all lowercase letters are turned to upper case, and all already uppercase letters remain as such, but, if these uppercase letters were not preceded by whitespace, they have prepended an underscore. Extending this, NaN stands simply for Not a Number, although, in JavaScript,

```
I: typeof NaN
O: "number"
```

And also,

```
I: NaN == NaN
O: False
```

Now, to weave things together, I have tracked down some dodgy JavaScript behaviour (today was otherwise mildly productive in implementing JavaScript GUI control functions):

```
var oldFacts = this.data;
this.data = {};
var numNewFacts = oldFacts.length - 1;
```

Where data is an array of Objects, yet, numNewFacts (sometimes) goes to NaN. Really, I plan on (and will now -- these entries, despite being reflective, are sometimes written concurrently with actions, but

they do not purport to be formal) printing the contents of all variables to the screen. Indeed, it must be found the cause for this error, not just a shoddy fix, and this will help find it. A snippet follows:

```
console.info("old -> ")
console.info(this.data)
console.info("old length ->")
console.info(this.data.length);
var factNumberToDelete = this.currentFactNumber;
_deselectFact(factNumberToDelete);
var oldFacts = this.data;
console.info("\"old\" length ->")
console.info(oldFacts.length);
this.data = {};
console.info("\"old\" length is still ->")
console.info(oldFacts.length);
var numNewFacts = oldFacts.length - 1;
console.info("numNewFacts -> ")
console.info(numNewFacts)
```

Experimentation with this structure yielded that, once a node has been removed, something terrible happens. After haphazardly pressing a few buttons, I realised something which I payed dedicated thought to earlier. Recently, I had just changed data from storing a list of strings to a list of Relation tuples, storing a description and a Relation constructor compatible Python-side. Either way, they main structure is an array -- why, then, do I write this.data = {}; ? Quite manifestly, the curly braces should be made square.

The instance of erroneous behaviour disappears, yet, the erroneous behaviour remains. I will speak little of it (in fear of writing more than doing), but I must expound, after an iteration of deletions, this.data contains a series of clones of the somethingth element of the old data array. It is plain where this error lies, and it can whence be fixed with relative ease.

In a few seconds (despite what the paragraphing might imply), I realised that JavaScript must not evaluate ternary operators in the order I projected it to, which is quite reasonable of them. I was clearly erring to write:

```
var factNumberToCopy = factNumber + (factNumber >= factNumberToDelete) ? 1
: 0;
```

Things still aren't working very well, but I must adjourn. The problems will likely be tacitly solved between now and next report.

1.37. COPYING
[22/06/2016]

When copying rich HTML (with CSS rules) into a contenteditable div, it retains its style on insertion. Looking this up yields http://stackoverflow.com/questions/12027137/javascript-trick-for-paste-as-plain-text-in-execcommand. I am not sure how to approach this problem, but it can be easily deferred for now (namely, I am not sure of the extent to which the problem must be fixed -- do I need to apply

changes to all areas which have the contenteditable tag?). Otherwise, the GUI is slowly but surely being developed, and much of the functionality in line with the Python code has been developed, but not yet linked.

It can be noticed that the JavaScript code is not internally documented to a great deal of quality, unlike the Python code. I feel I can justify that this is because it isn't really part of the solution. What I consider to be the solution is the functional, interesting, and testable Python code, which has an aim of solving a well defined problem. The JavaScript is merely a method of interaction between the user and the actual solution, and, taking a random snippet, how would it be had that I explain this:

```
PLE.facts.select = function (factNumberToSelect)
{
    _deselectFact(this.currentFactNumber);
    this.currentFactNumber = factNumberToSelect;
    var newSelectedNode =
        document.getElementsByClassName("fact")[factNumberToSelect];
    newSelectedNode.style.backgroundColor = "pink";
    document.getElementById("factDescription").innerText =
        this.getDescription();
};
```

(excuse the pink placeholder colour)

The final line, for instance, can seem mildly perplexing and out of place -- but it necessarily cannot warrant the comment "// since a new fact has been selected, the description of the current fact must be updated", and this quotation is the only possible descriptor for this functionality which adds any meaning. Tomorrow, I will work on bridging the Python and JavaScript (once I have added a quaesitum insertion point).

1.38. BRIDGING
[23/06/2016]

The click event that would usually have triggered a response for the selection of a Relation has been replaced with a mousedown event, making it impossible for the user to highlight text in a relation without causing correlating style updates.

I am having the usual CSS difficulties: things are broken. After some fiddling, things appear to be working. This doesn't instil a great deal of confidence, but with no necessity of browser compatibility, truly, the stylesheet will be sturdy irrespective of any internal fragility.

It is currently being undergone the bridging of JavaScript and Python, at which point, the GUI will be actually functional. Currently, the GUI can feed primitive data into the PLE, but the PLE cannot return results. Again, progression past this point is not limited by any particular problems (which is very rare in this project), but rather, a lack of time.

1.39. OPERATIONAL FUNCTIONALITY [24/06/2016]

A log for a test case of the system follows:

```
Running PLE, welcome!

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](1 - [0]) + [0](1 - [1])

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

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

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

The GUI allows the accomplishment of the same thing:

```
adding fact "[0](1 - [1]) + [1](1 - [0])"
getting request "[1]"
1[0]
adding fact "[0](1 - [1]) + [1](1 - [0])"
getting request "[0]"
1[1]
```

Things, slowly, are still progressing. It is left to do these few tasks:

- Implement symbol dictionaries.

- Add saving and restoring functionality.

- Print results to GUI rather than CLI.

1.40. KNOWN ERRORS
[25/06/2016]

Present in the solution currently, and perhaps even in distribution, is an error associated with the HTML renderer: https://bugs.chromium.org/p/chromium/issues/detail?id=290466. There is no fix other than a redesign -- which is doable, but the issue is of ridiculously low priority.

…

1.41. INITIAL USER FEEDBACK
[17/07/2016]

The involvement of users is of extreme importance in the design process -- in the end, it will be them who spend the most time with the system. Walking my brother through the system, several suggestions were made (of which some important ones I repeat):

- Colour coded way of seeing whether or not a fact is valid.

- Rather than having to manually equate a facts mathematical representation to 0 (eg. [0] = [1] becomes [0](1 - [1]) + [1](1 - [0]) = 0), have this task handled by the computerised aspect of the system.

- Descriptions are displayed on the hovering over of a fact, not just after selection.

According to these suggestions, I hastily implemented runCLI.__transpose(constructor) → transposedConstructor, and am fiddling with a colouring system.

1.42. TIME
[18/07/2016]

Without much difficulty, I finished colour coding Relations as being red or green for incorrect and correct respectively, and decided to implement a similar colour coding feature for symbols in superposition. But, for some time, I have known that there is no way I will get to implement all features I wish to (this was an initial expectation). Hover-text is an example of a low-priority feature that would be nice to have, but in the interest of eXterme Programming, its inclusion will certainly be neglected (along with many other desirable features).

…

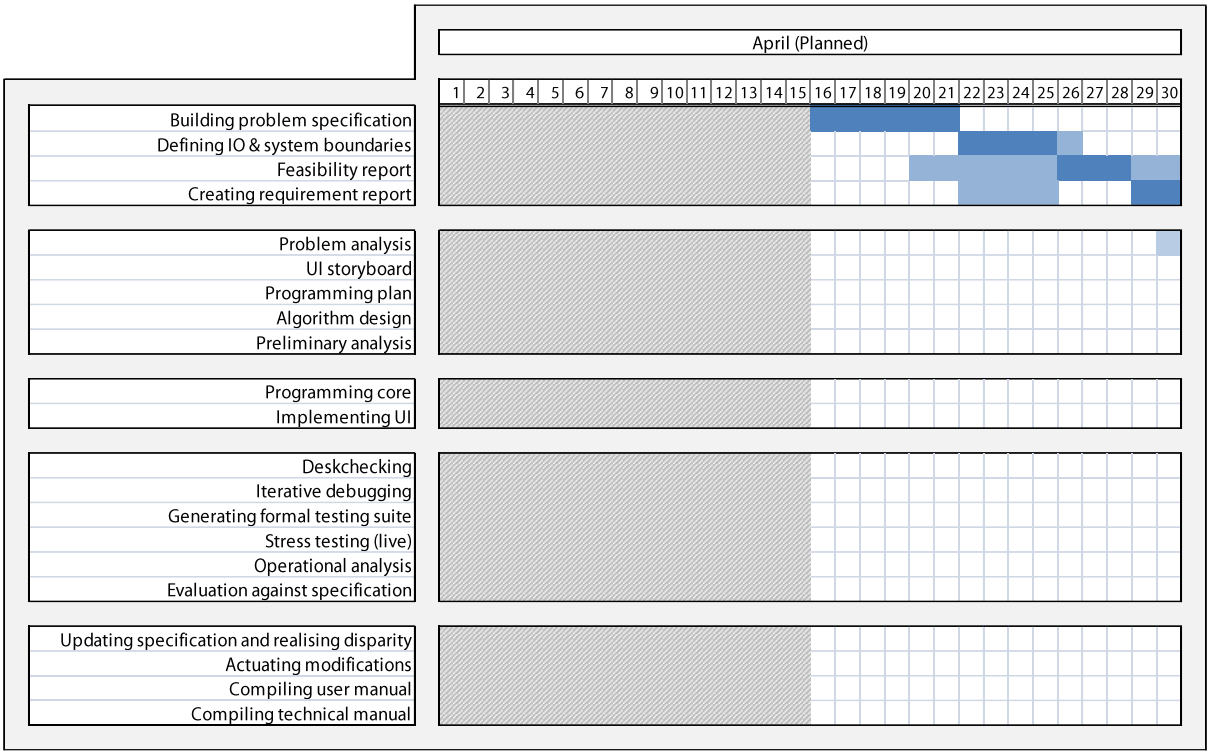
1.42. FINAL LOG (OVERTIME) -- AND ON THE LACK OF LOGS [31/07/2016]

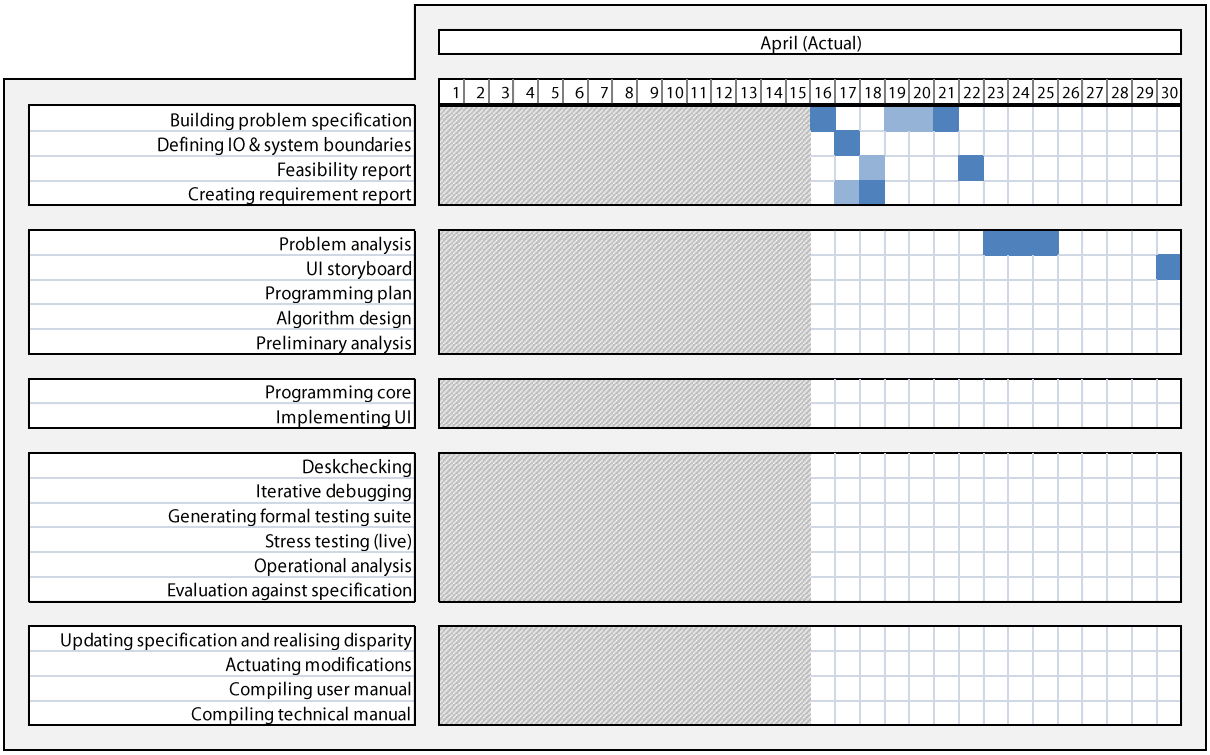
My final thoughts on the project find their place in the evaluation, not here, where the subject is on why there wasn't anything worth logging during the projects close. There was about a 20 day gap where I didn't really work on the project at all (apart from doing a fairly significant rewrite -- which is worth a mention, but there is not much to explain past that), and so, logs were not necessary. Then, on the 15th (July), I eased into iterating on the already functional program to make it releasable. During this period, there were a lot of small modifications amounting to a significant total change, but really, nothing particularly mentionable in writing that doesn't fit into the evaluation section. Towards the very end of the allocated time (over the past few days), I rushed to get a bunch of administration stuff done. This, I think, was because I would have liked to have written the admin content earlier, but I didn't want to do it in case I decided to change the program significantly, so I really thought it all through without writing anything, making it fairly quick when I was confident that no significant changes would be made.

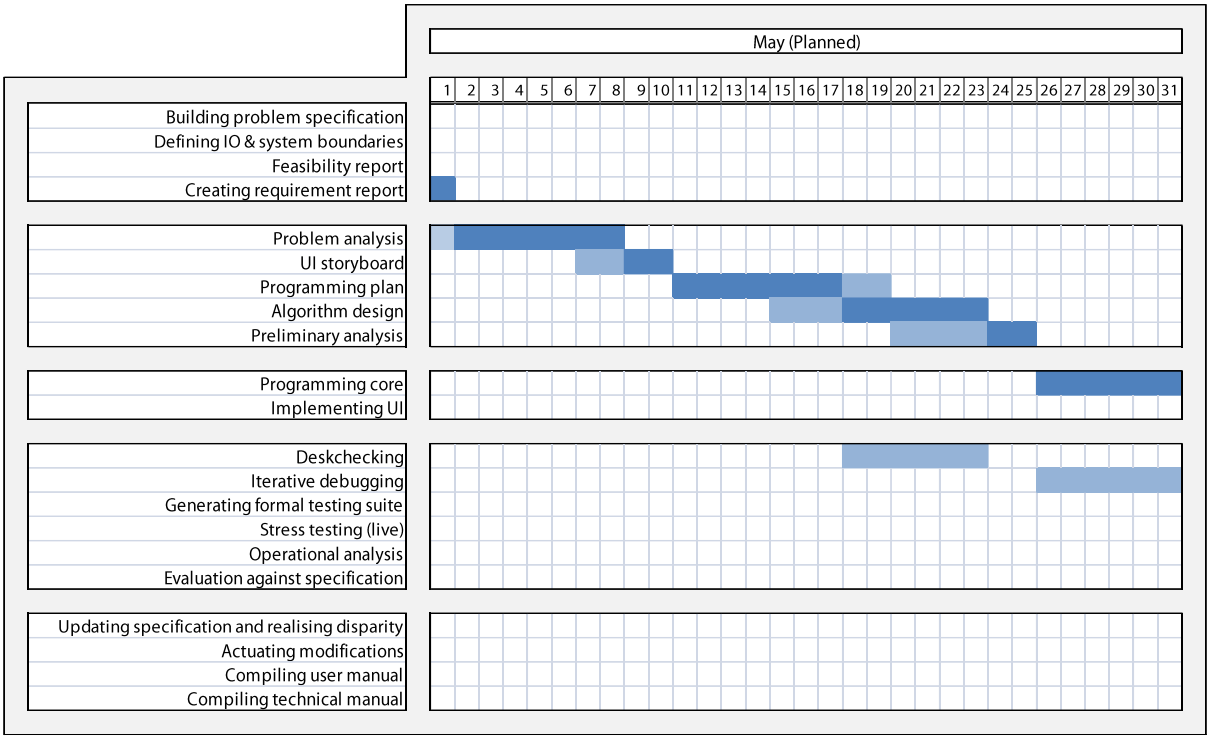
2. PROJECT TIMELINE

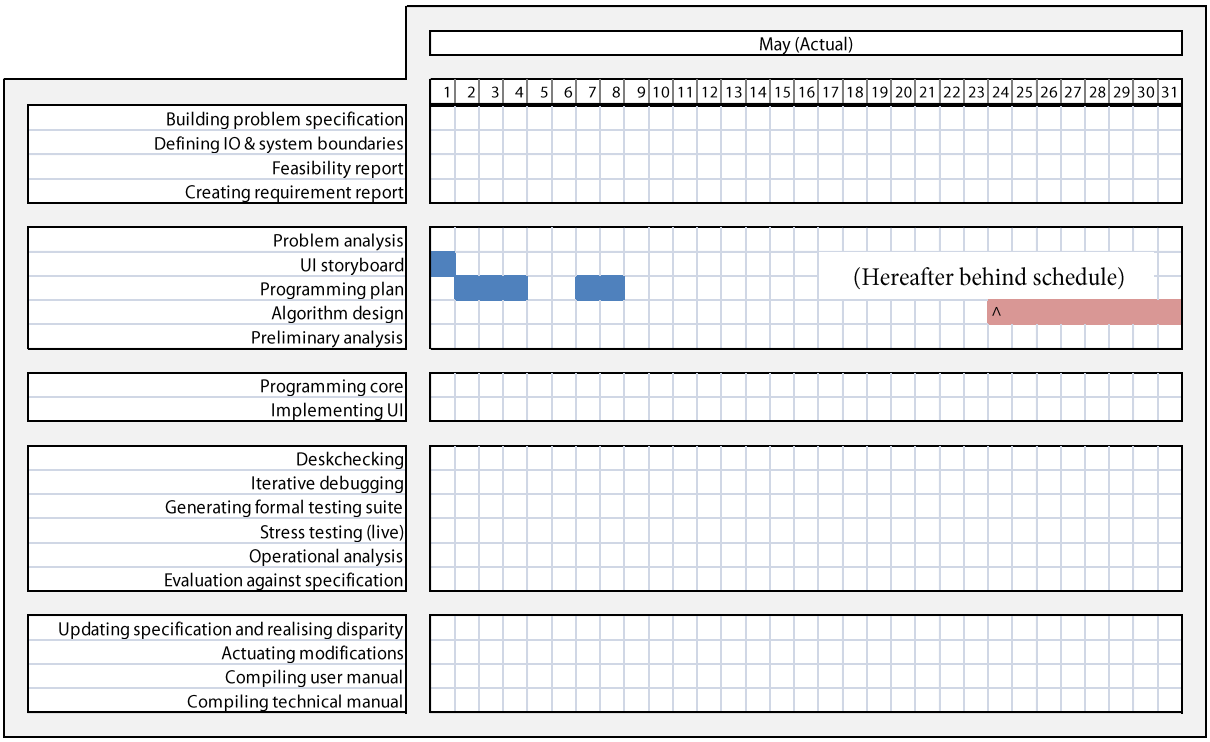
2.1. GANTT CHART

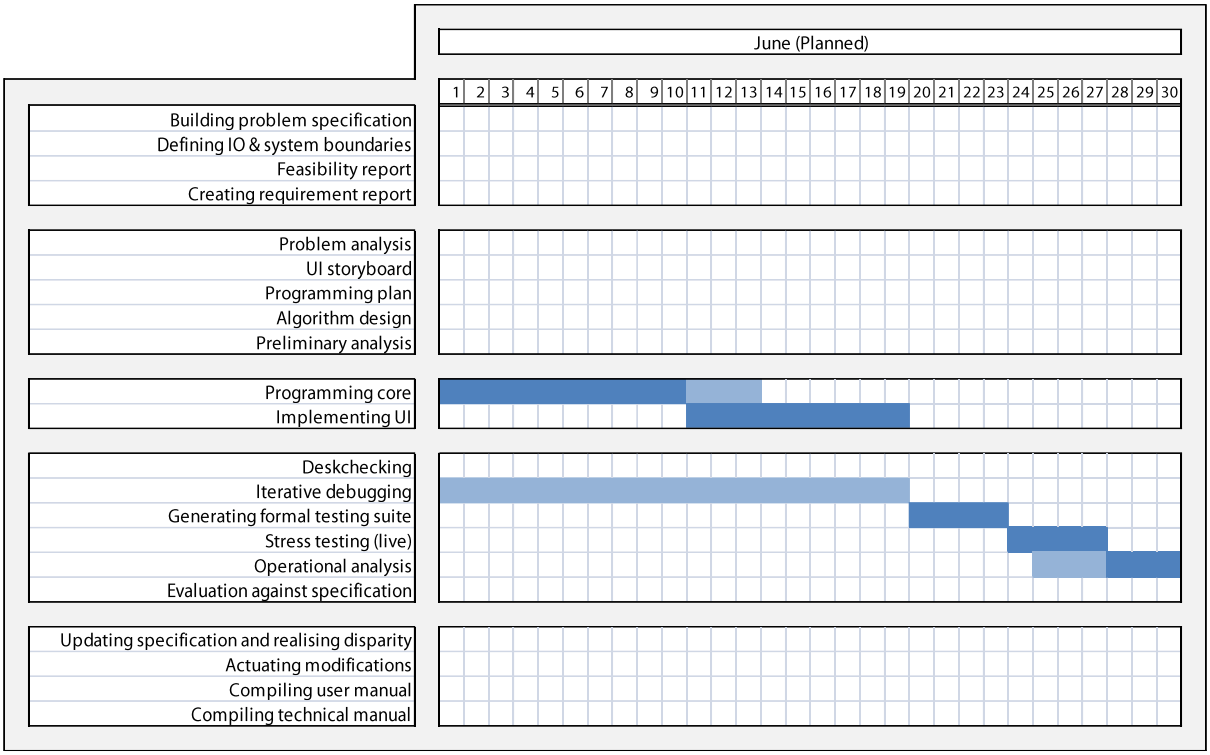
(Dark shades confer that a lot of effort be (or has been) put in, light shades express the converse) --

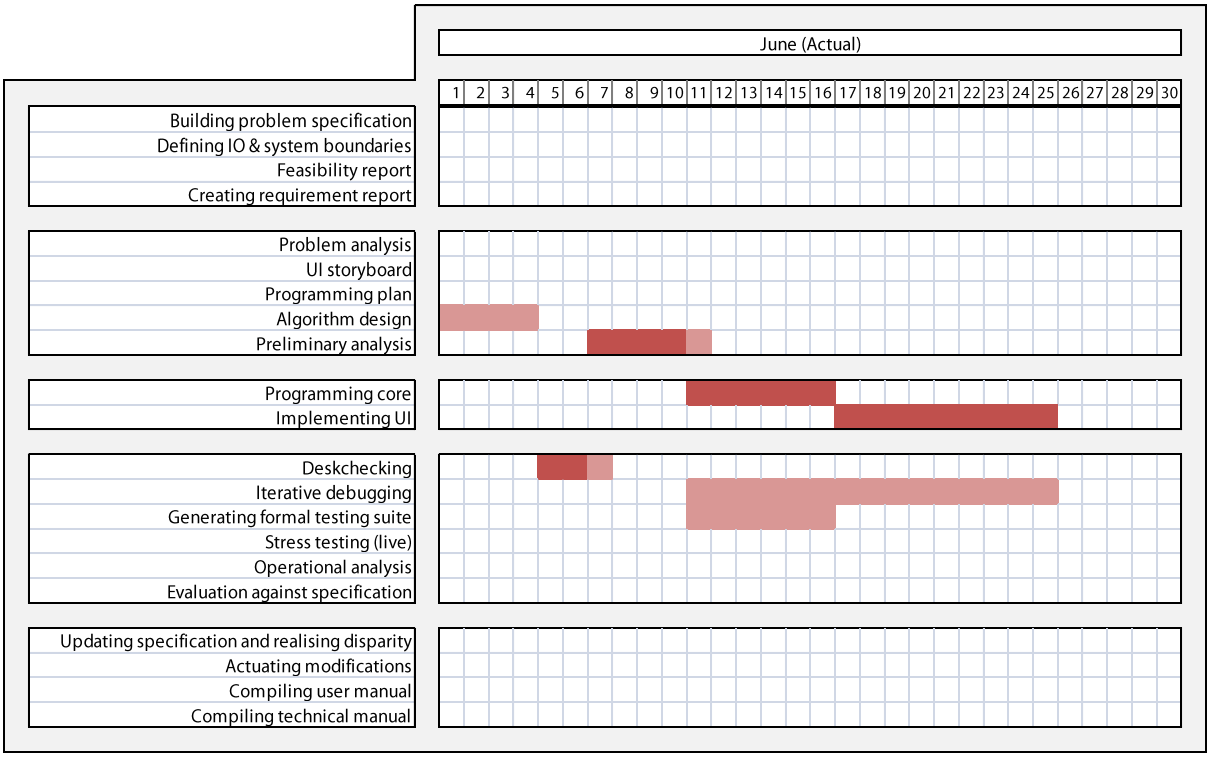


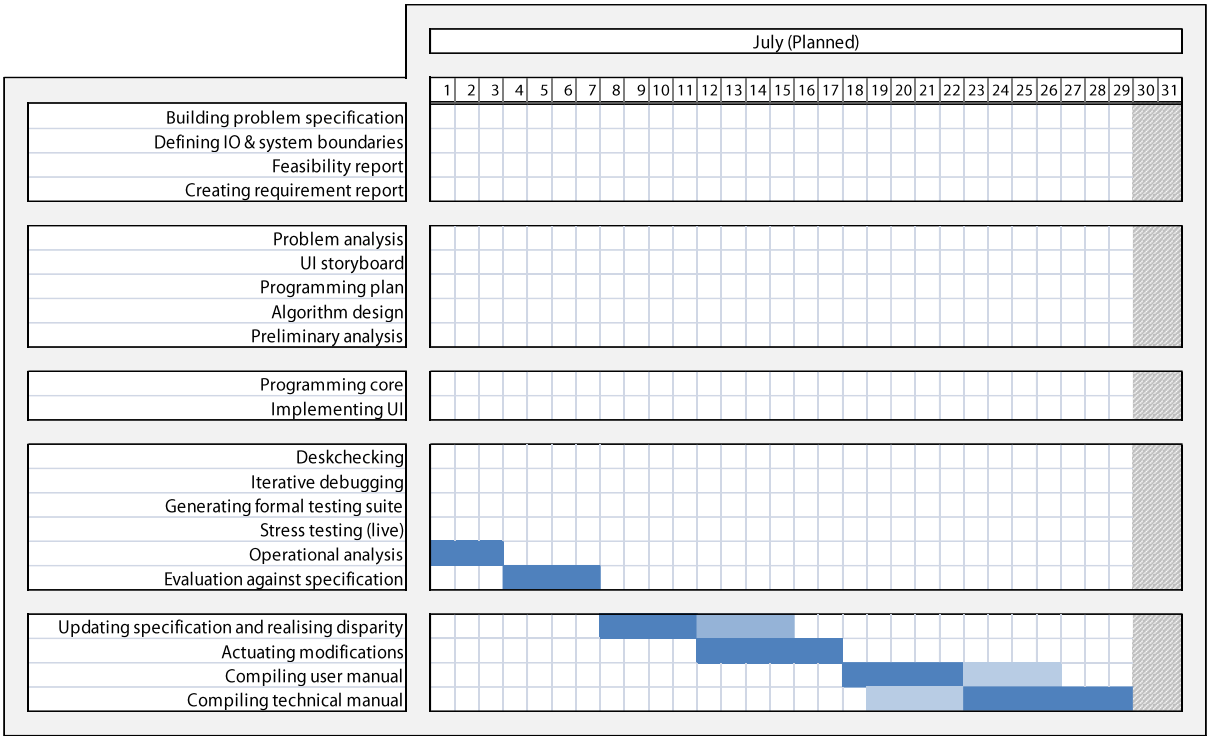


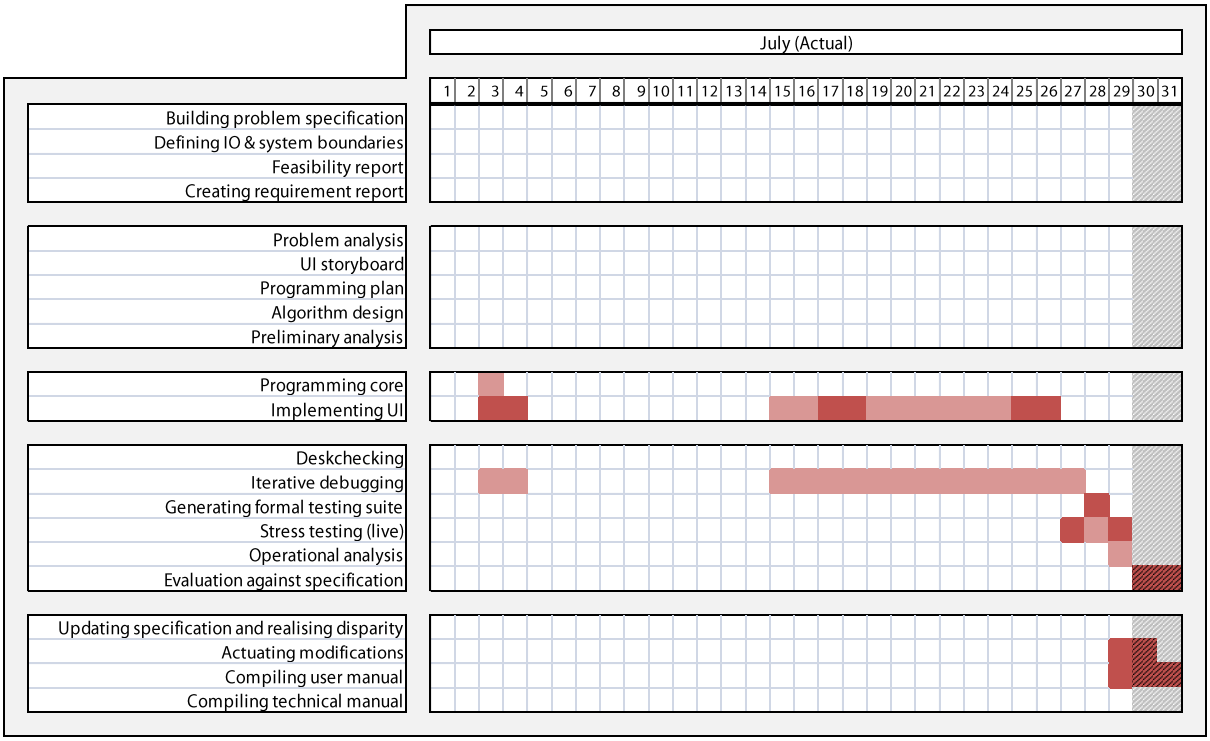

May (Planned)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Building problem specification
Defining IO & system boundaries
Feasibility report
Creating requirement report
Problem analysis
UI storyboard
Programming plan
Algorithm design
Preliminary analysis
Programming core
Implementing UI
Deskchecking
Iterative debugging
Generating formal testing suite
Stress testing (live)
Operational analysis
Evaluation against specification
Updating specification and realising disparity
Actuating modifications
Compiling user manual
Compiling technical manual


May (Actual)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Building problem specification
Defining IO & system boundaries
Feasibility report
Creating requirement report
Problem analysis
UI storyboard
Programming plan
Algorithm design
Preliminary analysis
(Hereafter behind schedule)
^
Programming core
Implementing UI
Deskchecking
Iterative debugging
Generating formal testing suite
Stress testing (live)
Operational analysis
Evaluation against specification
Updating specification and realising disparity
Actuating modifications
Compiling user manual
Compiling technical manual


June (Planned)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Building problem specification
Defining IO & system boundaries
Feasibility report
Creating requirement report
Problem analysis
UI storyboard
Programming plan
Algorithm design
Preliminary analysis
Programming core
Implementing UI
Deskchecking
Iterative debugging
Generating formal testing suite
Stress testing (live)
Operational analysis
Evaluation against specification
Updating specification and realising disparity
Actuating modifications
Compiling user manual
Compiling technical manual


June (Actual)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Building problem specification
Defining IO & system boundaries
Feasibility report
Creating requirement report
Problem analysis
UI storyboard
Programming plan
Algorithm design
Preliminary analysis
Programming core
Implementing UI
Deskchecking
Iterative debugging
Generating formal testing suite
Stress testing (live)
Operational analysis
Evaluation against specification
Updating specification and realising disparity
Actuating modifications
Compiling user manual
Compiling technical manual


July (Planned)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Building problem specification
Defining IO & system boundaries
Feasibility report
Creating requirement report
Problem analysis
UI storyboard
Programming plan
Algorithm design
Preliminary analysis
Programming core
Implementing UI
Deskchecking
Iterative debugging
Generating formal testing suite
Stress testing (live)
Operational analysis
Evaluation against specification
Updating specification and realising disparity
Actuating modifications
Compiling user manual
Compiling technical manual


July (Actual)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Building problem specification
Defining IO & system boundaries
Feasibility report
Creating requirement report
Problem analysis
UI storyboard
Programming plan
Algorithm design
Preliminary analysis
Programming core
Implementing UI
Deskchecking
Iterative debugging
Generating formal testing suite
Stress testing (live)
Operational analysis
Evaluation against specification
Updating specification and realising disparity
Actuating modifications
Compiling user manual
Compiling technical manual

2.2. TIME MANAGEMENT STRATEGIES

Time management is the most important aspect of project management, and is surely the primal cause of many problems and responses during software engineering. In accord, some strategies must be employed so as to allow the efficient management of scarce time resources. The successfulness of these strategies is the most important factor in determining the success or failure of the project.

1. Adopt a generalised method of eXtreme Programming. Programming methodologies are usually generalisable to a much broader scope (namely, any form of project management), but here, the important point is to *do something*. Don't simply plan (or plan to plan under this strange development scenario), create something that works and iterate upon it. This has connotations in that progress will be made continuously rather than all in one go, so if the development is halted unexpectedly, there will still be some product and little wasted planning. This strategy allows for less work to be wasted when the client (who ought to be involved heavily in the development process) invariably changes their mind about the scope of the solution, or when it is realised that some aspects of the solution are unachievable within the given timeframe. In summary: do things; don't just plan to do things. If there is sufficient time, iterate until the desired degree of quality is attained.

2. Don't be too ambitious. Software projects specifically are wont to fail for a multitude of reasons, so failure must be expected, and it will be a pleasant surprise if this fate is escaped. Knowledge of this means that features will be added sparingly under the approximate course of reasoning: the project is already doomed to fail; why should this perhaps unnecessary feature be added? This state of mind brings the amount of queued tasks well down, meaning that there is simply less to do.

3. Acknowledge that efficiency is a function of stress. Attempt must be made to continuously make tangible progress in the project, and nothing must be left to the last minute. It may be noticed that the Gantt chart in `I. 2.1. has a deadline of the 30th, while the actual deadline is not until the 5th of the next month. This is primarily to avoid stress, thus relieving detriment to efficiency.

4. Most importantly: make plans for time management -- schedule reasonable amounts of work. The schedule should be fitted to the worst possible case of success for the project, namely, tasks should stretch equally from the start and end date of the project (excluding the buffer that was mentioned prior -- this is a backup). In this way, a Gantt chart can be incrementally updated and compared to the planned Gantt chart, and if ever progress falls behind that which was planned, it can be plainly said "If I keep going at this rate, the project will fail". This makes it clear when corrective action must be taken, such as decreasing the workload, or increasing the work rate. The sum implementation of these strategies will lessen the likelihood of failure of the project -- and less sceptically -- lead to success.

DEFINING & UNDERSTANDING THE PROBLEM

PART II.

1. SPECIFICATION OF THE PROBLEM

1.1. ASSERTING THE NEED FOR A PROPOSITIONAL LOGIC ENGINE

Studying Law, Matthew Killeen tacitly takes interest in the construction of logical arguments, and is inclined to be dependent on propositional logic to a somewhat outlandish extent (relative to average). The formal universe of applying symbolic to Law has been termed *Jurimetrics* by Lee Loevinger in 1940. This study has applications that are mostly concerned with the overlaying of pure logic and statistics to individual questions of law, that is to say, answering questions of individual cases. Applications of AI to Law cover topics including:

- Indexed information retrieval,

- Legal text classification and summarisation,

- Information extraction from databases and texts,

- Models of evidential legal reasoning,

- Models of argumentation and decision making,--

Of which needs, the last two are fitted by a propositional logic engine.

It is perhaps a schismatic topic of how Law and morals are related. The simple question of "what is Law" makes a division majorly threefold under the philosophies of Natural Law, Positive Law, and Critical Legal Studies. Without even the need for exploration of these views, it becomes sufficiently obvious that dilemmas exist not only within Law (as Jurimetrics primarily deals with), but about Law also. It follows with little question that argument is requisite in the domain of Law, perhaps unnervingly to a layperson. An environment of such conflict, by its very nature, births a demand for arguments of higher quality. Argument assistance quickly becomes of much help in both the study and practice of Law, and can here be classified as a need, thus revealing a problem to which a solution can be developed and applied.

The need for "argument assistance" -- a propositional logic engine -- does not exist merely in Law; the subject serves simply as a convenient stone to tie the issue down to, and must be the end to which functionality is provided due to its apparent extremism. The study of Law is where the issue warrants a solution, and it follows that a student of Law is to be the client,-- although the solution will still provide much help to the other disciplines, and users from both categories must necessarily be considered equally as the primary audience. More concisely and clearly, the following assertions must be made:

1. A propositional logic engine (P.L.E) is of moderate use to particular divisions of a mass audience, who can be considered as end users of a P.L.E.

2. The demands of a mass audience are uninstantiated; a P.L.E is not needed by a mass audience (to the extent of a warranted solution), but it would see some use.

3. Lawyering is a domain in which a P.L.E is of such a help that it can be considered a need.

4. Law is only a niche area where a P.L.E, if developed, is applicable.

Common reason and uncomplicated values conducts us to the important results that, since the client should be an embodiment of the need, Matthew Killeen should be the primary candidate for the client, but it will be an aggregation of him and people external to the study of Law who make up the group of end users.

1.2. PRELIMINARY SURVEY OF NEEDS

Speculatively, a P.L.E has applications in computer engineering (particularly prominent in software engineering), sciences, mathematics, philosophical conundrums, as well, of course, as Law, and many other more specific cases (which exist, perhaps, also within the disciplines mentioned). With an audience manifold, any person can be plucked out of hyperspace and they will have experienced a scenario where a P.L.E would have been of some convenience, but they do not necessarily know what propositional logic is, so the scope of questions is somewhat limited so as to avoid the bias of inserting a causatively necessary brochure or summary of what propositional logic is. A simple survey follows:

1. (T / F) I have heard of the term "propositional logic" outside of this survey.

2. (T / F) I have heard of the term "syllogism" outside of this survey.

3. (T / F) I have had some experience drawing a truth table or a Venn diagram.

4. (T / F) I was aware prior to this survey that the process of drawing conclusions from a set of facts is mechanisable.

5. (Range) I am commonly faced with tasks where I must derive logical conclusions from some database of facts.

6. (Range) I am commonly faced with tasks where I must determine the truth of a logical conclusion given a set of propositions.

7. (Range) My experience places the process of reaching or arguing logical conclusions as being complex.

8. (Range) Computerised tools have assisted me in reaching logical conclusions.

Putting this poll to an abstract audience, the following conclusions can be drawn after reviewing results from each respective question:

1. Everyone who has heard of propositional logic has also heard of syllogism -- thus, the responses to this question can be swiftly disregarded in summary.

2. Sixty-three percent of the surveyed populace has heard of either syllogism or propositional logic (which can be treated as synonymously), thus, the selected audience are explicitly aware of the domain within which the solution will operate.

3. All surveyed members have drawn either truth tables or Venn diagrams, and thus, inherently recognise some form of manual solution to different extents of logical problems.

4. The division within the selected demographic with respect to the explicit awareness of the automatability of syllogism is absolutely parallel.

5. There is an approximately equal division relevant to the frequency at which propositional logic is employed by the responders in terms of deriving logical conclusions.

6. Solution verification is a process that occurs in approximately the same fifty-fifty division as arriving at the conclusions. The extent of the verification and deriving of facts correlate for each responder: on average, there is a seven-point-five percentage point difference between an individual person's response to questions 5 and 6, thus, arriving at and verifying logical conclusions are done under similar circumstances.

7. The process of reaching or arguing logical conclusions is placed as being somewhat complex (obtaining sixty-three percent average complexity).

8. Computerised tools have not been helpful to the audience, averaging less than one-fifth agreement (of extent) with the helpfulness of current computerised solutions.

The perhaps tacit hypothesis, thus, can be verified: it is true that conclusions reached through propositional logic are oftentimes required by abstract members of a populace, the process of which is known to be mechanisable, and is seen as difficult, yet, few mechanised solutions that would aid in the process are used -- interpretably because high quality solutions may not exist.

One additional inference should be made about the requisite functionality of the program (although this was not the aim of this round of surveys), and it concerns the correlation of responses between 5 and 6. The correlation implies that the faculties of proof and deriving logical conclusions are equally desirable by any individual potential user, and whence, the solution should offer dual functionality in this regard.

(An appended note for the last paragraph): It has since come to the attention of the writer that this behaviour is easily simulatable without particular addition of features (verily, the solution is functionally complete). For example, if a user wishes to check whether Relation A is equivalent to Relation B, they can open two sessions and compare results. If they wish to inquire into whether or not Relation A implies Relation B, they can open a session with just Relation A, and then another

session with both Relation A and B, and compare results. This comes very intuitively, and ought to have been recognised earlier.

1.3. EXCERPTS FROM CLIENT INTERVIEW

The client's responses to the following interview questions accentuate the advantage that stands to be gained through the application of propositional logic to law, and instantiate types of scenarios where the solution can be applied. The responses establish the client's perceptions of what the problem is, and certainly, this perspective must be iteratively acknowledged throughout the development process.

(i) Law is complicated. How so?

> "When we think of law, we often think of a set of rules that should apply to everyone in a community. But we fail to realise that all law is expressed in words, and all words can have ambiguity. People use and apply these laws differently. Social and technological advances occur such that laws operate in ways that the original lawmakers did not intend. People make decisions, not computers, and these people bring their own views as to public policy and justice to the podium of the law. In this way, the law changes. The law may be very clear in its operation, but that operation may create a manifestly absurd or injustice outcome. How then should the court deal with that? But it is not nearly this simple. Whilst all of these changes occur, judges remain adamant in holding the very opposite of the truth; they must present the law in such a way that it appears as though it never changes. It is tradition that lawyers and judges hold that new law (that is, new common law) is not new law, but rather newly discovered law. The tradition of hiding new law whilst also making it, combined with inherent ambiguity and human biases is what makes the laws complicated."

(ii) You commonly encounter dilemmas to which propositional logic is favourable over strenuous thought. What are some examples of these dilemmas?

> "Law is concerned with logic, but it seldom engages the serious interest of logicians, largely for the reasons aforementioned in answer (i). It was famously said by Lord Coke that 'law is artificial reasoning'. Whilst this is true to the extent that there is uncertainty of outcomes, there is much that can be modelled using propositional logic.
>
> To understand the practical operations of logic in the law, we should consider something in the law called the fact/law distinction.
>
> There are two features present in any case, they may be called fact and law. The facts are (traditionally) found by a jury, and pertain to the circumstances and conduct that amounted to the situation that the court is concerned with.
>
> The law is the set of rules (although not really rules) that is applied to the circumstances found in fact. In theory this is significant for, once fact is found, it is not usually subject to appeal. Furthermore, it is the decisions of judges in law and not in fact that bind future judges whereas the facts are held to be

different in each circumstance. Again, there is no bright line rule here either. In practice many of the seemingly factual decisions can be framed as legal questions by skilled lawyers.

This makes a propositional logic engine a useful tool for apply facts (a circumstance that is ascertained by document discovery, interviews and investigation) to law (that which is ascertained by legal research). In this way, propositional logic can render an answer which is how the law would be applied. But we must appreciate that law is an evolving instrument, even though it gives the appearance that it is not. Policy considerations (the way the law *ought to be*, and not the way the law *is*) often shape judgements. So we must seek to use propositional logic in such a way that it informs us how the law has worked historically, but not necessarily presently.

Recognising this limitation, we may seek to use an engine only to inform us which cases to proceed with, or what the likely outcome might have been, but certainly it should not tell us how to argue a case.

We might also be interested in applications beyond cases, for instance in the many ethical dilemmas that arise in the practice of law. Ethical questions are not asked at every point during the pursuit of a case, for it would be menial to do so. That said, increased scrutiny of process would be valuable, and such an engine might achieve that whilst being resource-efficient."

2. REQUIREMENT REPORT

2.1. BOUNDARIES

(i) The Propositional Logic Engine (P.L.E) will not perform lexical analysis; all information must be inputted in mathematical form, and will be outputted also in mathematical form.

(ii) The P.L.E will not process any problems of quantity or probabilities: the only quantifiers the P.L.E will interpret are *some*, *none*, and *all*.

(iii) The P.L.E allows for the aided interpretation of a solution, it does not, itself, wholly provide the solution.

(iv) The P.L.E operates in the universe of *propositional logic*, that is, all results inferred from outputs or intermediary equations of the P.L.E will be equally as fallible as the premises combined, whence:

(v) The P.L.E will not yield objectively true solutions.

(vi) The P.L.E will not be able to derive implied relations between symbols. The system will only act on explicitly provided information, or information stored in a database.

2.2. SPECIFICATION

(i) The functionality of the P.L.E must be: given a relationship between symbols, accept and parse a request for knowledge on a subject, and return the knowledge the system has on the requested subject. The P.L.E must interface in this format of exchange:

1. User says "*These* facts are true".
2. User asks "What is true about *this* subject?"
3. The P.L.E fulfils the request, saying "*These* facts are true", --

Input	Process	Output
Rules (relationships between symbols), Request for information on a subject	Conglomerate all given rules, then search and compile information relevant to the requested subject.	Information about the requested subject.

(ii) The P.L.E must accept and parse multiple relations.

(iii) The P.L.E must have saving functionality for symbol's names and the relationships between symbols.

(iv) The main process of the P.L.E must be manually parallelisable.

(v) The P.L.E must conform reasonably to the following quality criteria by the perspective of all parties involved:

- correct and accurate
- reliable
- usable
- efficient
- secure

(vi) The P.L.E must conform to the following criteria by the developer's perspective:

- interoperable, flexible, and reusable
- testable
- maintainable
- possess high quality wholly

(vii) The adding of additional features, if it is to occur at all, will be done so in consultation with the client, and will have their own specifications developed externally to this section (perhaps tacitly).

3. FEASIBILITY REPORT

3.1. TIME & NATURE OF THE PROBLEM

By the factor *time* is meant the ratio of effort required to produce the solution to the timeframe over which the production is to take place. The timeframe is fixed; the solution is invariably required for distribution by Friday 5th August 2016. The magnitude of work that must be done to produce the solution is however indeterminable, rendering all feasibility with respect to time as being speculative. With little experience, it follows that the speculation is wild hearsay, but to assert it as being perhaps incorrect is to nullify some of the harm it being incorrect may cause. The summary, further, is hindered in detail by the fact that the reader and writer have no common experience on project work, so for it to be said "this is a medium sized project" is entirely meaningless without shared ground.

The Gantt chart in `I. 2.1., by experience, is easily accomplishable and is quite generous in terms of time allocation. A little over three months is plenty of time to complete it well beyond satisfaction of myself and the client. `I. 2.2. contains time management strategies that detail how success can be asserted. Further, the complexity of the problem isn't high, and the problem is of an easily solvable nature, offering a smorgasbord of paths for possible solutions, where the quality of the solution can be traded off in self-contained packets for time. If the solution becomes unexpectedly difficult, time still should not be an issue, because although the timeframe is fixed, the amount of time spent working on the project within the timeframe is variable, and can increase ridiculously if it is required (at the cost of things external to the project). In whole, the solution is very solvable within the timeframe.

3.2. DEVELOPER SKILL AND COMPETENCY

The developer has had experience in the development of programs somewhat manifold, and has had a light education in the domain of computer engineering; therefore there is little doubt about his ability to produce such a simple solution. The developer's programming knowledge is complemented by knowledge of the subject of propositional logic.

The developer is confident to solve the program in a variety of different programming languages, and is confident that, if it is required, additional languages or libraries will willingly be learnt. It may be true that there are languages appropriate to the logical paradigm, but they are not necessarily more favourable as they are surely less versatile, and it is the developer's preference to use a language in the domain which he has had experience with, or is comparable to a language which he holds experience in.

3.3. PERFORMANCE

The program will run invariably at $\sim O(2^n)$, so performance is a major issue. $O(2^n)$ is the theoretical minimum as by the unsolved *exponential time hypothesis*, and since this is not an academic work, it is not understandable that the time complexity falls below $O(2^n)$, and space complexity will, without optimisation, suffer similarly. It is imperative thus that the process is designed to be parallelisable, but

past that, there need not be much concern under performance. The process is a natural, repetitive one, and the automated solution will tend to be much faster than a manual solution. There is much room for optimisation, but this is easily secondary, as any sort of performance issue lies explicitly in the magnitude inputs, not in the program.

3.4. RESOURCES

The issue of resources ought not to be prevalent in this instance of software development, due particularly to its simplicity and relatively small size. Resources in terms of understanding the problem are many and various, resources in terms of hardware are assumed to be met due to their lack of demand, but resources in terms of software is slightly more complex. The developer, with enough time, does not require any library add-ons or IDE's, but in the finite-time scenario, they are very valued. As a student (and by the benevolence of Microsoft), the developer has available to them a multitude of IDE's. Free libraries and extensions are also all that should be required due to the low-level nature of the problem.

3.5. INTEROPERABILITY

The feasibility of the project, wholly, is only really threatened by considerations of client hardware. The primary environment details follow:

```
[System Summary]
OS Name                             Microsoft Windows 7 (Enterprise)
Version                             6.1.7601 Service Pack 1 Build 7601
System Type                         x64-based PC
Processor                           Intel(R) Core(TM) i5-4670 CPU
                                          @3.40GHz, 3401 Mhz, 4 Core(s),
                                          4 Logical Processor(s)
Locale                              Australia
Time Zone                           AUS Eastern Standard Time
Installed Physical Memory (RAM)     8.00 GB
Total Physical Memory               7.91 GB
Available Physical Memory           5.65 GB
Total Virtual Memory                15.8 GB
Available Virtual Memory            13.5 GB
Page File Space                     7.91 GB

[Display]
Name                                Intel(R) HD Graphics 4600
Adapter RAM                         (2,080,374,784) bytes
Resolution                          1920 x 1080 x 60 hertz

[Disks]
Drive                               C:
Media Type                          Fixed hard disk
Size                                465.76 GB (500,105,736,192 bytes)
```

But all efforts must be made to increase the interoperability of the system. The program must consider the amount of RAM and hard-disk space available on the individual machine it runs on so as to avoid

running out of memory or space, which is a quite realistic issue with a problem of exponential nature. There, however, are no theoretical minimums, and optimisation can help remedy the issue.

Interoperability is certainly feasible, but it is perhaps hard to achieve to a desirable extent. This issue must be further explored later.

(An appended note -- feasibility is not revisited): the program is fully dependent on Python support on the machine. Past terrible surprises, the PLE *theoretically* works on any machine that complies with Python standards.

3.6. SUMMARY OF FEASIBILITY

The achievement of a high quality solution appears now to be quite realistic and feasible. The primary points of concern are with the availability of easy to use GUI libraries and resources, which can lead to the development requiring more dedicated work within the timeframe and perhaps some suffering under the "usability" quality descriptor; secondarily, an issue could easily be found in terms of interoperability and performance if optimisations and other failsafe considerations are not appropriately made, and parallelisation is unable to be implemented but deemed necessary -- but there is no foreseeable reason as to why this would happen, and the undertaking can be concluded as quite sensible.

4. SOCIAL & ETHICAL CONSIDERATIONS

4.1. A GENERALISED EXISTENCE OF SOCIAL & ETHICAL ISSUES

Social and ethical issues inexorably exist in all software projects, but their probabilistic occurrence groups their existence into particular categories of solutions. The problem of propositional logic is not especially assailable by any such category of issues, but still, an acknowledgement is facilitated in infinite scepticism of the future.

Firstly it must be considered the negative impacts of the solution on society. The solution does not nearly replace human involvement in any form, but offers assistance in the deduction of logical conclusions in combination with human operation. The results of the system carry very much the requirement of external interpretation. I cannot conceive of a time in which the P.L.E removes the necessity of people from a system, or of a time in which the solution invokes the capacity of malign operations. Any concerns of this nature are also made to diminish by the limited distribution of the system.

The next concern is pertaining to the actual development of the solution, namely, does the solution avoid issues of intellectual property. By the exact nature of development, most questions of authorship are nullified, but in the event that aspects of other people's works are incorporated, it must necessarily be considered. At this moment, it can at best be asserted that source code attached to the project have one sole developer, however, the distributed solution will have authors ranging from the owners of used libraries, the language implementer, the creators of tools used in development, and many others.

Most pragmatically comes the expounding of any concerns regarding inclusivity. Indeed, with any program, it must be made true that a multitude of audiences, regardless of their aspect, are able to operate it. The means must be offered to allow the solution to be usable by absolutely different demographics of people, but it does not necessarily fall to the sole developer (especially in the case of limited distribution) to consider every individual demographic. In the development of the standard GUI, many simple adjustments can be made to broadly extend the inclusivity of the program, but, complete interface redesigns would be required to extend usability amongst many domains. It is thus that the programs source will be available so as to allow the inclusivity of the system to be extended to particular domains as by others needs (in purchasable products, an interface would have to be developed).

PLANNING & DESIGNING THE SOLUTION

PART III.

1. PROBLEM ANALYSIS

1.1. NATURE OF THE PROBLEM

It is difficult to summarise the functionality and correctness of Boolean algebra without oversimplification (or ground-up abstract algebra), but the most important thing to recognise is that it has one perceived difference with regular algebra, and that point of difference is division. The disparity is not because of the shortcomings of Boolean algebra, but due to common misconceptions in standard algebra -- in difference, the Boolean statement $\frac{theta}{theta}$ cannot be simplified to unity. In this section, many axioms are assumed inherently to be true, but a few definitions are warranted:

(a)	$\Theta \neq \emptyset$;
(b)	$\forall \theta_n \in \Theta \; \exists \; \theta_{n+1} \in \Theta$;
(c)	$\forall \theta_n \in \Theta \; \exists \; \theta_{n-1} \in \Theta$;
(d)	$\forall \theta_n \in \Theta . \; \theta_n \cdot (1 - \theta_n) \equiv 0$;
(e)	$\Delta \neq \emptyset$;
(f)	$\forall \delta_n \in \Delta \; \exists \; \delta_{n+1} \in \Delta$;
(g)	$\forall \delta_n \in \Delta \; \exists \; \delta_{n-1} \in \Delta$;
(h)	$\forall \delta_n \in \Delta . \; \delta_n \cdot (1 - \delta_n) \equiv 0$;
(i)	$\forall \delta_n \in \Delta \; \exists \; \varphi(\delta_n) \subseteq \Theta$;
(j)	$\forall \theta_n \in \Theta, \delta_n \in \Delta . \; \theta_n \in \varphi(\delta_n(\theta_n)) \leftrightarrow \delta_n(0) \neq \delta_n(1)$;

Less formally:

(a)	THETA is not an empty set
(b)	For every member of THETA, there exists a member with a greater index. These members are denoted by theta.
(c)	THETA is an infinite class, and its members are denoted by $theta_n$ where $\lvert n \rvert < \infty$.
(d)	Members of THETA ($theta_n$) are subjected to the law of non-contradiction, that is, $theta_n$ can take the form of either 0 or 1, but not both at once (unless unobserved).
(e)	DETLA is not an empty set.
(f)	For every member of DETLA, there exists a member with a greater index. These members are denoted by delta.

(g)	DELTA is an infinite class, and its members are denoted by $delta_n$ where $\|n\| < \infty$.
(h)	Members of DELTA ($delta_n$) are subjected to the law of non-contradiction, that is, $delta_n$ can evaluate to the form of either 0 or 1, but not both at once (unless unobserved).
(i)	Members of DELTA ($delta_n$) functions have THETA type parameters / arguments.
(j)	Phi is a function that returns all of the "important" parameters of DELTA type, that is, all parameters that cause a different result when given the arguments 1 and 0.

In adherence with the IPO diagram in the specification `II. 2.2., the general equation can be represented in a purely mathematical form. The derivation of what this form means now follows.

From II. 2.2.,

$q = f(r, F_0, F_1, \dots F_n);$ (1)

Where:

$f \rightarrow$ an arbitrary relation, in this case, the process which defines the program,

q → quaesitum (that which is sought after),

r → request,

$F_0, F_1, \dots F_k \rightarrow$ facts.

By both r and F_n, let it be meant a proposition connecting two Boolean functions, for that is indeed what constitutes either a fact or request:

$r, F_n \rightarrow \delta_p = \delta_q;$

The subject and predicate can be transposed and squared:

$r, F_n \rightarrow (\delta_p - \delta_q)^2 = 0;$

Noting the rules initially furnished, $\delta^2 = \delta$, thus

$r, F_n \rightarrow \delta_p - \delta_q\delta_p + \delta_q - \delta_q\delta_p = 0;$

$\therefore r, F_n \rightarrow \delta_p(1 - \delta_q) + \delta_q(1 - \delta_p) = 0;$ (2)

It can here be noticed that $\varphi(r)$, $\varphi(F_n) \subseteq \varphi(\delta_p) \cup \varphi(\delta_q)$, which will play part in the operational efficiency of the algorithm (as well as impacting memory usage).

In order to solve a series of facts for a request, the facts must be conglomerated. Let this be represented by V, then,

$V = F_0 + F_1(1 - F_0) + F_n(1 - F_{n-1})\dots(1 - F_1)(1 - F_0);$

$V = \sum_{n=0}^{k} F_n \cdot \prod_{m=0}^{n-1} (1 - F_m);$ (3)

In terms of factorisation, it is worth noting that nesting can be generalised from the form of five symbols: $((1 - F_0)((1 - F_1)((1 - F_2)((1 - F_3)(F_4) + F_3) + F_2) + F_1) + F_0)$.

Combining (2) and (3), letting
i) $P = \delta_p$, the subject,
ii) $Q = \delta_q$, the predicate, then
$V = P_0(1 - Q_0) + Q_0(1 - P_0) + [P_1(1 - Q_1) + Q_1(1 - P_1)][\ P_0Q_0 + (1 - P_0)(1 - Q_0)] + \ldots$
$+ [P_n(1 - Q_n) + Q_n(1 - P_n)][\ P_{n-1}Q_{n-1} + (1 - P_{n-1})(1 - Q_{n-1})]\ldots[P_1Q_1 + (1 - P_1)(1 - Q_1)]$
$[P_0Q_0 + (1 - P_0)(1 - Q_0)];$

It can be deduced thus from this general structure of V, regardless of the subjects and predicates of the facts (the values of the argument), V is of type delta, and will yield a type theta result if called with all arguments, otherwise, will return a lambda function of type delta.

The fundamental problem (1) can importantly be restated as:
$q = f(r, V);$ (4)

It is required that V is developed with respect to the symbols of r, $\varphi(r)$, which are of type theta. Noting that V is of type delta and is still restricted to the Boolean domain, the generalisation next developed can be applied, where:
$G(\delta, s) \rightarrow$ development of δ with respect to the symbols s
$\delta \rightarrow$ relation of type delta,
s → set of symbols of type θ, individually labelled with subscripts, --

The function G can be defined by its behaviour, assuming $\varphi(\delta) = (\theta_0, \theta_1, \ldots \theta_n)$:
$G(\delta, \emptyset) = \delta(\theta_0, \theta_1, \ldots \theta_n);$ …
$G(\delta, \{\theta_a\}) = \theta_a\delta(\theta_0, \theta_1, \ldots, \theta_{a-1}, 1, \theta_{a+1}, \ldots\ \theta_n) + (1 - \theta_a)\delta(\theta_0, \theta_1, \ldots, \theta_{a-1}, 0, \theta_{a+1}, \ldots\ \theta_n);$ …
$G(\delta, \{\theta_a, \theta\}) = G(G(\delta, \{\theta_a\}), \{\theta\});$ (5)

It is clear to see that the number of constituents grows exponentially, and this is where the complexity of the problem is founded. The complexity is $O(2^n)$ to the exponent, n, defined as being $\max(|\varphi(\delta_p)|, |\varphi(\delta_q)|)$, although it later grows to $|\varphi(\delta_p) \cap \varphi(\delta_q)|$.

All developments, $G(\delta, s)$, take the form of a sum of a series of products of pairs of constituents and coefficients, that is to say:

$$G(\delta(\theta_a, \theta_b, \theta_c), \{\theta_a, \theta_b\}) = \delta(1, 1, \theta_c) \cdot \theta_a\theta_b + \delta(1, 0, \theta_c) \cdot \theta_a(1 - \theta_b) + \delta(0, 1, \theta_c) \cdot (1 - \theta_a)\theta_b + \delta(0, 0, \theta_c) \cdot (1 - \theta_a)(1 - \theta_b);$$

The constituents are respectively $\theta_a\theta_b$, $\theta_a(1 - \theta_b)$, $(1 - \theta_a)\theta_b$, $(1 - \theta_a)(1 - \theta_b)$;
The coefficients are respectively $\delta(1, 1, \theta_c)$, $\delta(1, 0, \theta_c)$, $\delta(0, 1, \theta_c)$, $\delta(0, 0, \theta_c)$;
Coefficients can take either a theta or delta form: theta under the condition of "full development", $G(\delta, \varphi(\delta))$, and otherwise, they can only be assumed as delta.

The General Solution from George Boole's "An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities", 1854, IX. METHDS OF ABBREVIATION. 9., can be furnished in detail as follows, so as to define how the abstract function *f* solves for its first argument in terms of its second (4).

$\delta_v \rightarrow$ aggregation of all knowledge
$\delta_t \rightarrow$ sought expression

Letting $\varphi(\delta_t) = \{\theta_0, \theta_1\}$ so as to allow for generalisation.

Let δ_v be in the form demonstrated in (5), $G(\delta_v, \varphi(\delta_v)) = 0$;
$G(\delta_v, \varphi(\delta_t)) = \delta_a\theta_0\theta_1 + \delta_b\theta_0(1 - \theta_1) + \delta_c(1 - \theta_0)\theta_1 + \delta_d(1 - \theta_0)(1 - \theta_1)$;
$\delta_a\theta_0\theta_1 + \delta_b\theta_0(1 - \theta_1) + \delta_c(1 - \theta_0)\theta_1 + \delta_d(1 - \theta_0)(1 - \theta_1) = 0$; (6)

Let δ_t be in the similar form (5), $\theta_t = G(\delta_t, \varphi(\delta_t))$;
$G(\delta_t, \varphi(\delta_t)) = \theta_a\theta_0\theta_1 + \theta_b\theta_0(1 - \theta_1) + \theta_c(1 - \theta_0)\theta_1 + \theta_d(1 - \theta_0)(1 - \theta_1)$;
$\theta_t = \theta_a\theta_0\theta_1 + \theta_b\theta_0(1 - \theta_1) + \theta_c(1 - \theta_0)\theta_1 + \theta_d(1 - \theta_0)(1 - \theta_1)$; (7)

Transposing and squaring (7),
$\theta_t(1 - G(\delta_t, \varphi(\delta_t))) + (1 - \theta_t)G(\delta_t, \varphi(\delta_t)) = 0$; (8)

But
$1 - G(\delta_t, \varphi(\delta_t)) = 1 - \theta_a\theta_0\theta_1 + \theta_b\theta_0(1 - \theta_1) + \theta_c(1 - \theta_0)\theta_1 + \theta_d(1 - \theta_0)(1 - \theta_1)$;
$1 - G(\delta_t, \varphi(\delta_t)) = \theta_0\theta_1 + \theta_0(1 - \theta_1) + (1 - \theta_0)\theta_1 + (1 - \theta_0)(1 - \theta_1) - \theta_a\theta_0\theta_1 - \theta_b\theta_0(1 - \theta_1) - \theta_c(1 - \theta_0)\theta_1 - \theta_d(1 - \theta_0)(1 - \theta_1)$;
$1 - G(\delta_t, \varphi(\delta_t)) = (1 - \theta_a)\theta_0\theta_1 + (1 - \theta_b)\theta_0(1 - \theta_1) + (1 - \theta_c)(1 - \theta_0)\theta_1 + (1 - \theta_d)(1 - \theta_0)(1 - \theta_1)$; (9)

So reducing (8) with (9),
$\theta_t[(1 - \theta_a)\theta_0\theta_1 + (1 - \theta_b)\theta_0(1 - \theta_1) + (1 - \theta_c)(1 - \theta_0)\theta_1 + (1 - \theta_d)(1 - \theta_0)(1 - \theta_1)] + \ldots$
$(1 - \theta_t)[\theta_a\theta_0\theta_1 + \theta_b\theta_0(1 - \theta_1) + \theta_c(1 - \theta_0)\theta_1 + \theta_d(1 - \theta_0)(1 - \theta_1)] = 0$; (10)

(6) + (10) will represent the expression assuming the truth of the query,
$\delta_a\theta_0\theta_1 + \delta_b\theta_0(1 - \theta_1) + \delta_c(1 - \theta_0)\theta_1 + \delta_d(1 - \theta_0)(1 - \theta_1) + \theta_t[(1 - \theta_a)\theta_0\theta_1 + (1 - \theta_b)\theta_0(1 - \theta_1) + (1 - \theta_c)(1 - \theta_0)\theta_1 + (1 - \theta_d)(1 - \theta_0)(1 - \theta_1)] + (1 - \theta_t)[\theta_a\theta_0\theta_1 + \theta_b\theta_0(1 - \theta_1) + \theta_c(1 - \theta_0)\theta_1 + \theta_d(1 - \theta_0)(1 - \theta_1)] = 0$;
$\theta_0\theta_1[\delta_a + \theta_t(1 - \theta_a) + (1 - \theta_t)\theta_a] + \theta_0(1 - \theta_1)[\delta_b + \theta_t(1 - \theta_b) + (1 - \theta_t)\theta_b] + \ldots$
$(1 - \theta_0)\theta_1[\delta_c + \theta_t(1 - \theta_c) + (1 - \theta_t)\theta_c] + (1 - \theta_0)(1 - \theta_1)[\delta_d + \theta_t(1 - \theta_d) + (1 - \theta_t)\theta_d] = 0$; (11)

Which, mapping each constituent to a value in (11),
$C_a = \theta_0\theta_1$;
$C_b = \theta_0(1 - \theta_1)$;
$C_c = (1 - \theta_0)\theta_1$;
$C_d = (1 - \theta_0)(1 - \theta_1)$;
$\therefore 0 = \Sigma|C_n[\delta_n + \theta_t(1 - \theta_n) + (1 - \theta_t)\theta_n]|$, for $n = \{a, b, c, d\}$; (12)

Now let (12) be abstracted to some expression

$\delta_a\theta_0\theta_1 + \delta_b\theta_0(1 - \theta_1) + \delta_c(1 - \theta_0)\theta_1 + \delta_d(1 - \theta_0)(1 - \theta_1);$ (13)

Then the result of the elimination of $\varphi(\delta_t)$ from (13) will be:

$\delta_\delta(\theta_1, \theta_2) = \delta_a\theta_0\theta_1 + \delta_b\theta_0(1 - \theta_1) + \delta_c(1 - \theta_0)\theta_1 + \delta_d(1 - \theta_0)(1 - \theta_1);$

$\delta_\delta(0, \theta_2)f(1, \theta_2) = [\delta_c\theta_1 + \delta_d(1 - \theta_1)][\delta_a\theta_1 + \delta_b(1 - \theta_1)];$

$\delta_\delta(0, 0)f(1, 0)f(0, 1)f(1, 1) = [\delta_d][\delta_b][\delta_c][\delta_a];$ (14)

The parameters of (12) can now be reduced with respect to (14), and the total knowledge of the system represented in a more general form of (11):

$0 = \Pi|\delta_n + \theta_t(1 - \theta_n) + (1 - \theta_t)\theta_n|$, for $n = \{a, b, c, d\};$ (15)

Now, the sought expression will be the result of solving (15) for θ_t. Let (15) be represented in the form:

$0 = \Pi|\delta_\pi(\theta_t)|$, for $n = \{a, b, c, d\};$ (16)

$0 = \theta_t\delta_\pi(1) + (1 - \theta_t)\delta_\Pi(0);$

$0 = \theta_t[\delta_\pi(1) - \delta_\pi(0)] + \delta_\pi(0);$

$\theta_t = \delta_\pi(0) / [\delta_\pi(0) - \delta_\pi(1)];$ (17)

Equating (15) to (16),

$\Pi|\delta_\pi(\theta_t)| = \Pi|\delta_n + \theta_t(1 - \theta_n) + (1 - \theta_t)\theta_n|$, for $n = \{a, b, c, d\};$

$\therefore \delta_\pi(0) = \delta_n + \theta_n;$ (18)

$\& \delta_\pi(1) = \delta_n + (1 - \theta_n);$ (19)

Observing each case from (18), considering the nature of products,

$\delta_\pi(0)(\delta_n, \theta_n) = \delta_n + \theta_n$:

$\delta_\pi(0)(0, 0) = 0;$

$\delta_\pi(0)(0, 1) = 1;$

$\delta_\pi(0)(1, 0) = 1;$

$\delta_\pi(0)(1, 1) = 2;$

$\therefore \delta_\pi(0)$ goes to 0 if and only if $\delta_n = 0$ and $\theta_n = 0.$ (20)

Observing each case from (19) as above,

$\delta_\pi(1)(\delta_n, \theta_n) = \delta_n + (1 - \theta_n)$:

$\delta_\pi(1)(0, 0) = 1;$

$\delta_\pi(1)(0, 1) = 0;$

$\delta_\pi(1)(1, 0) = 2;$

$\delta_\pi(1)(1, 1) = 1);$

$\therefore \delta_\pi(1)$ goes to 0 if and only if $\delta_n = 0$ and $\theta_n = 1.$ (21)

From (20) and (21):

Let $E' = \Pi|\delta_\pi(0)|$, for $n = \{a, b, c, d\};$

Let $E = \Pi|\delta_\pi(1)|$, for $n = \{a, b, c, d\};$

From (11), for n = {a, b, c, d} implies for all $\varphi(\delta_t)$;

Let t equal the sought symbol θ_t, then by (17),
$t = E' / (E' - E);$ (22)

But $E' = \delta_\pi(0)$ is reduced to unity under all conditions but when $\delta_n = 0$ and $\theta_n = 0$, And $E = \delta_\pi(1)$ is unity unless $\delta_n = 0$ and $\theta_n = 1$,

But from (6), δ_n is the nth coefficient of the development $G(\delta_v, \varphi(\delta_t))$,
And from (10), θ_n is the nth coefficient of the development $G(\delta_t, \varphi(\delta_t))$,

∴ E' = Product of all coefficients in the development $G(\delta_v, \varphi(\delta_t))$ who, in the development of $G(\delta_t, \varphi(\delta_t))$, vanish.
& E = Product of all coefficients in $G(\delta_v, \varphi(\delta_t))$ who, in the development of $G(\delta_t, \varphi(\delta_t))$, are unity.

Returning now to the original equation, the formula $t = E' / (E' - E)$ will be the definition of f.

Recalling (1), (4), and applying the result above, f can now be defined as the mechanical process to map $[r, F_0, F_1, \ldots F_n]$ to [t, E, E'] with the intermediary [r, V], and send t to $E' / (E' - E)$, allocating the quaesitum t as q.

Past this, it is best to represent the system physically, yet still, a mathematical model can be drawn out.

Achieving the form (4) is mathematically trivial, so let the process start at
$q = f(r, V);$
Where r is of type delta = 0,
And V is of type delta = 0.

Let the product of coefficients in the full development who, in the sought symbol's development, are equal to 1, be represented as
$\prod_{i=1}^{2^{|\varphi(r)|}} E(r, V, i);$ (23)

Let the product of coefficients in the full development who, in the sought symbol's development, are equal to 0, be represented as
$\prod_{i=1}^{2^{|\varphi(r)|}} E'(r, V, i);$ (24)

From (23), (24), applying (22),

$$q = \frac{\prod_{i=1}^{2^{|\varphi(r)|}} E'(r, V, i)}{\prod_{i=1}^{2^{|\varphi(r)|}} E'(r, V, i) - \prod_{i=1}^{2^{|\varphi(r)|}} E(r, V, i)} \quad (25)$$

And it remains only that the functions E and E' are defined.
From (5), let each constituent of $G(\delta, s)$ have a unique subscript, $G(\delta, s)_n$, then
$E(r, V, i) = G(r, \varphi(r))_i + (1 - G(r, \varphi(r))_i)G(V, \varphi(r))_i;$

$E'(r, V, i) = (1 - G(r, \varphi(r))_i) + G(r, \varphi(r))_i G(V, \varphi(r))_i;$

$$q = \frac{\prod_{i=1}^{2^{|\varphi(r)|}} \left(1 - G\left(r, \varphi(r)\right)_i\right) + G\left(r, \varphi(r)\right)_i G\left(V, \varphi(r)\right)_i}{\prod_{i=1}^{2^{|\varphi(r)|}} (1 - G\left(r, \varphi(r)\right)_i) + G\left(r, \varphi(r)\right)_i G\left(V, \varphi(r)\right)_i - \prod_{i=1}^{2^{|\varphi(r)|}} G(r, \varphi(r))_i + \left(1 - G\left(r, \varphi(r)\right)_i\right) G\left(V, \varphi(r)\right)_i};$$

And more particularly, what is sought takes the adorned form $G(q, \varphi(q))$ which is a development with coefficient types $\frac{\text{theta}}{\text{theta}}$, so the full mechanical equation is:

$$q = G\left(\frac{\prod_{i=1}^{2^{|\varphi(r)|}} \left(1 - G\left(r, \varphi(r)\right)_i\right) + G\left(r, \varphi(r)\right)_i G\left(V, \varphi(r)\right)_i}{\prod_{i=1}^{2^{|\varphi(r)|}} (1 - G\left(r, \varphi(r)\right)_i) + G\left(r, \varphi(r)\right)_i G\left(V, \varphi(r)\right)_i - \prod_{i=1}^{2^{|\varphi(r)|}} G(r, \varphi(r))_i + \left(1 - G\left(r, \varphi(r)\right)_i\right) G\left(V, \varphi(r)\right)_i}, \varphi(V) \cap \varphi(r)^C\right);$$

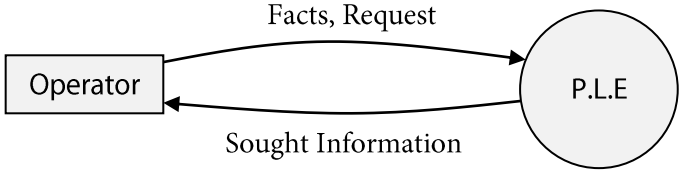
This simplified form must intrinsically be recognised in order to solve the problem, although, it is not necessary that the results can be explicitly derived. In summary, the program must form one master relation which equates all of the facts of the system to 0, and then, this must be developed with respect to the symbols in the sought relation. The result then takes the form of the development of E' / (E' - E), where E is the product of coefficients in the full development who, in the symbol's development, are equal to 1, and E' is the product of coefficients in the full development whose respective coefficients in the sought symbol's development are in it for 0.

1.2. NATURE OF THE SOLUTION -- FORMAL MODELS

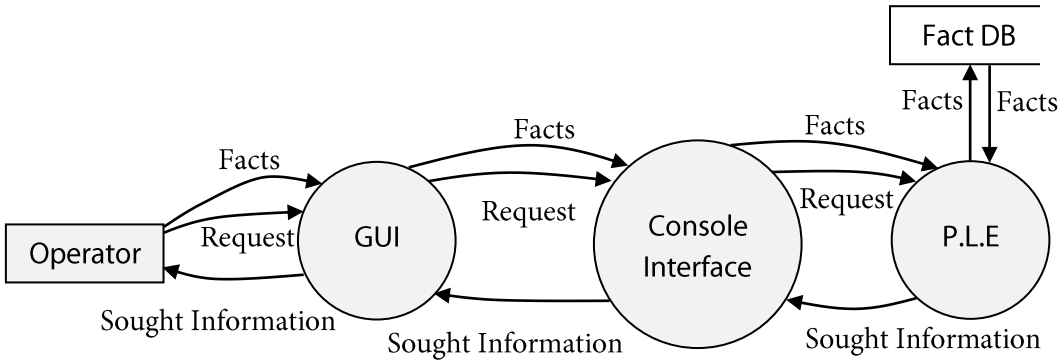
Defining the nature of the problem allows its exact analysis, and a solution to be developed so as to model the problem. A refined IPO diagram can now be made as an extension to the mathematical premise:

Input	Process	Output
1) Series of paired subjects and predicates constituting the facts of the system. 2) Request for information 'r', also in the form of a paired subject and predicate.	1) Aggregate the facts to form 'V'. 2) Develop 'V' with respect to all the symbols in 'r'. 3) Fully develop 'r' with respect to all of its symbols. 4) Select from the development of 'V' all coefficients of the constituents who, in 'r', are equal to 1, and assign the product to E. 5) Select from the development of 'V' all coefficients of the constituents who, in 'r', are equal to 0, and assign the product to E'. 6) Let Ω = E' / (E' - E), where 'Ω' is the unrefined quaesitum. 7) Develop 'Ω' fully -- with respect to all of its symbols -- and assign the result to 'q'.	1) Quaesitum 'q', which is all information about the requested subject as provided by the series of facts.

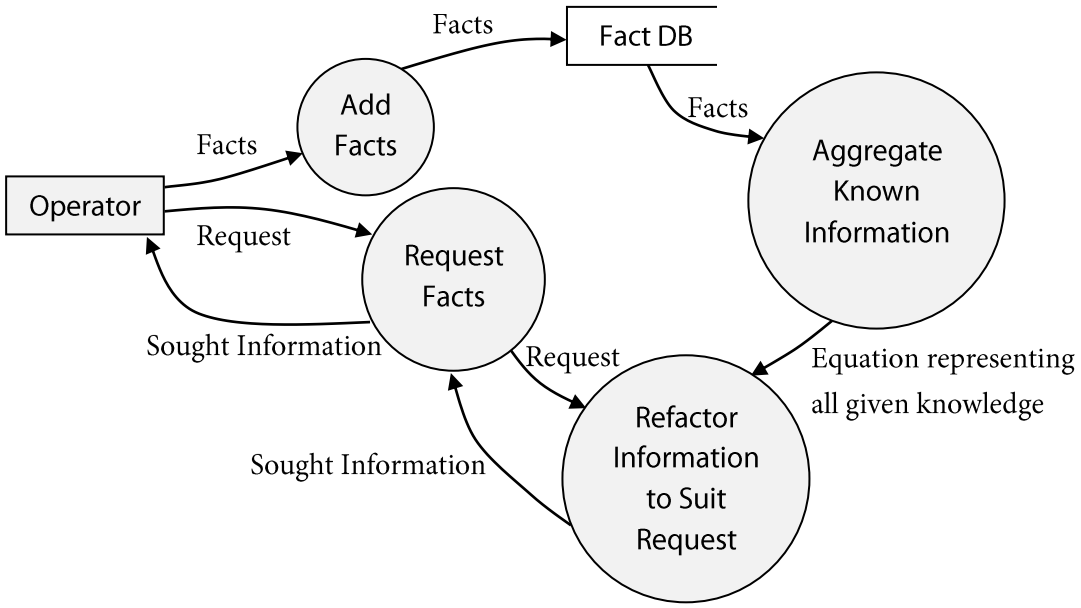
The context of the system happens to exist in the simplest possible permutation:



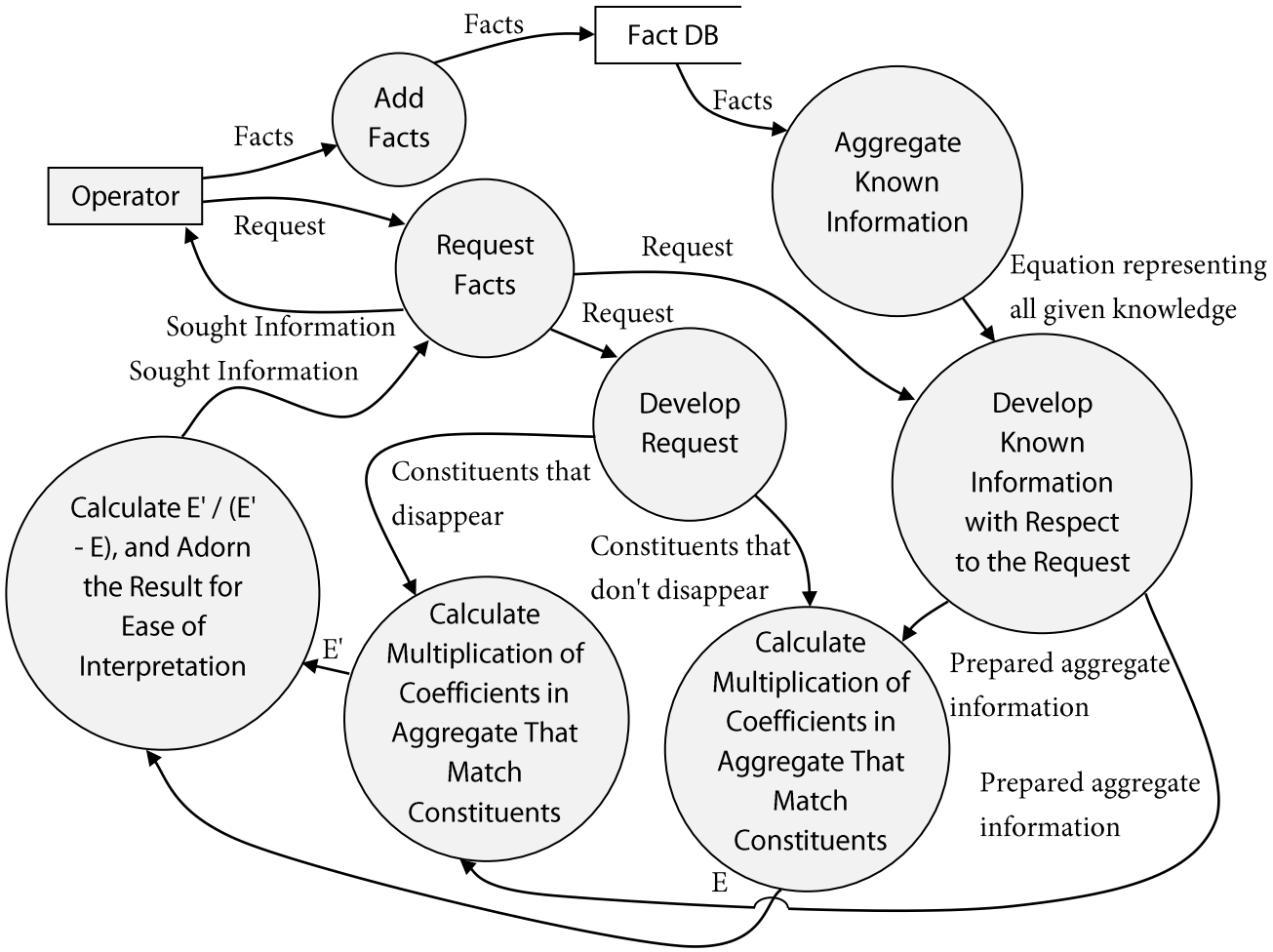
-- The operator can be either a human, using either the GUI or console interface, or a computer, which is restricted to the console interface. Refinement of the context diagram into a low level DFD yields:



This representation, however, is not stylistically correct. The *GUI* and *Console Interface* processes, firstly, do not begin with a verb. Most appropriately, they would begin with "pass", making "Pass through GUI" and "Pass through Console Interface" considering they do not do anything other than pass information between other processes (which is the exact nature of interfaces). Further -- since the processes don't do anything -- they are redundant, and really shouldn't be shown at all if this diagram is to labelled as legitimate, although it benefits that they are shown somewhere so that the interface of the system is best understood, and so that it can be demonstrated that the GUI is an entirely separate process to the actual propositional logic engine. Correcting and refining:



Further refinements are forced to revert to the more complicated mathematical roots of the problem. The *Refactor Information to Suit Request* process is the only one that warrants modularisation, but the whole DFD ought to be reprinted for completeness:



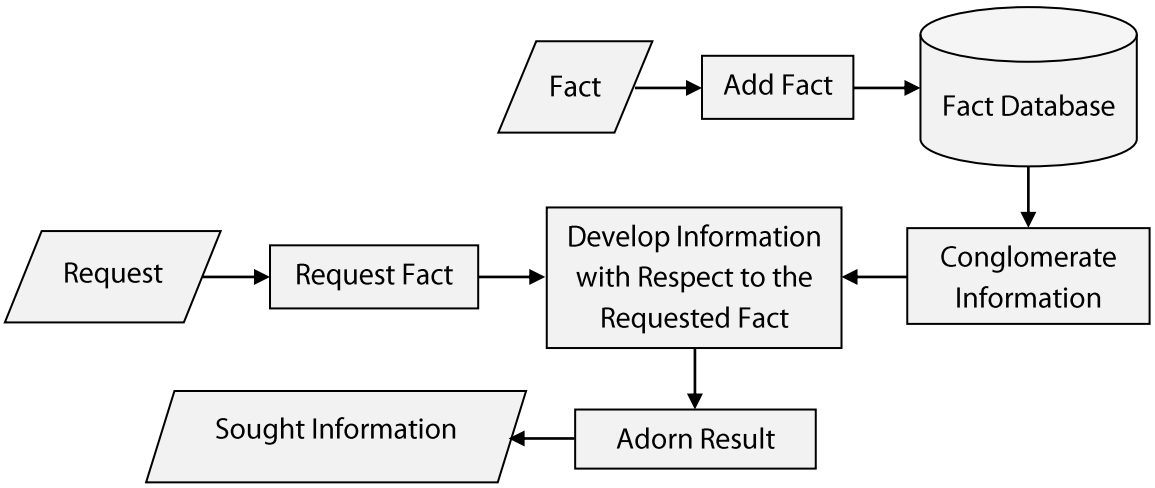
As a direct result of the program being mathematical in nature, the solution must have implemented the means to perform mathematics. This means that the solution must recognise the relationship between symbols, coefficients, terms, operators, expressions, and relations; either explicitly or implicitly. Most languages provide a great deal of mathematical functionality, many even an *eval* function, which would -- if used correctly -- allow the evaluation of mathematical expressions as strings, rather than as dedicated classes. The solution must recognise (highlighting means that the character is to be treated as a term in the EBNF, rather than an operator):

```
Positive Integer = {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

Symbol           = [<Positive Integer>]

Coefficient      = [-] <Positive Integer>

Term             = [<Coefficient>]{<Symbol>}

Operator         = + | - | * | /

Expression       = (<Term> {<Operator> <Expression>}) | (( <Term> {<Operator> <Expression>} ))
```

Relation	= <Expression> = <Expression>

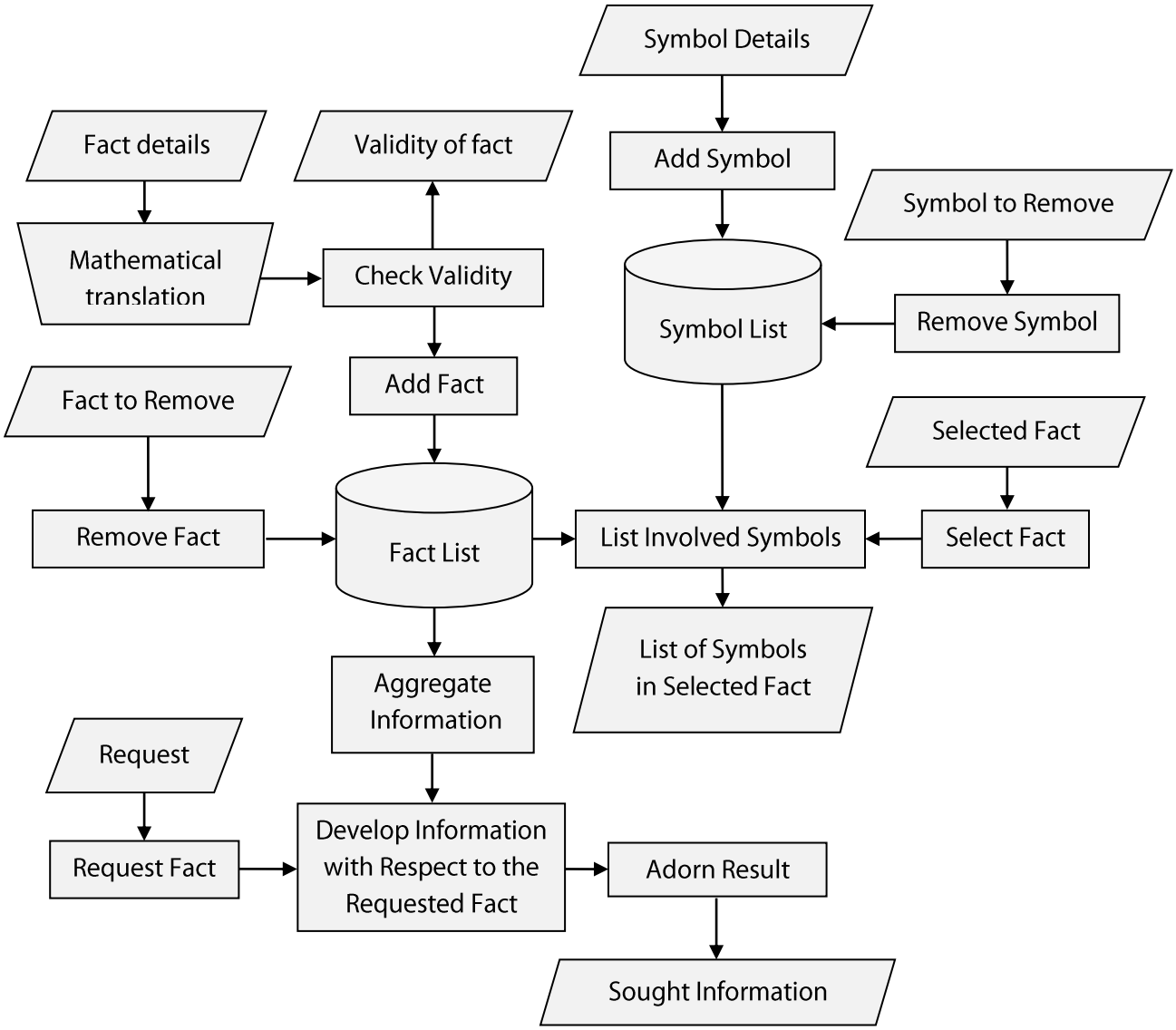
It must be noticed that there must be an infinite number of 'Symbols' -- otherwise, symbols such as letters would eventually run out. As aforementioned, these relationships must be somehow recognised by the program, namely, there must be a Relation data structure with which to conveniently store and manipulate algebraic equations.

Against the external world, the solution can be modelled quite simplistically as follows, and therein is conferred how the solution must interface with the operator and store data:



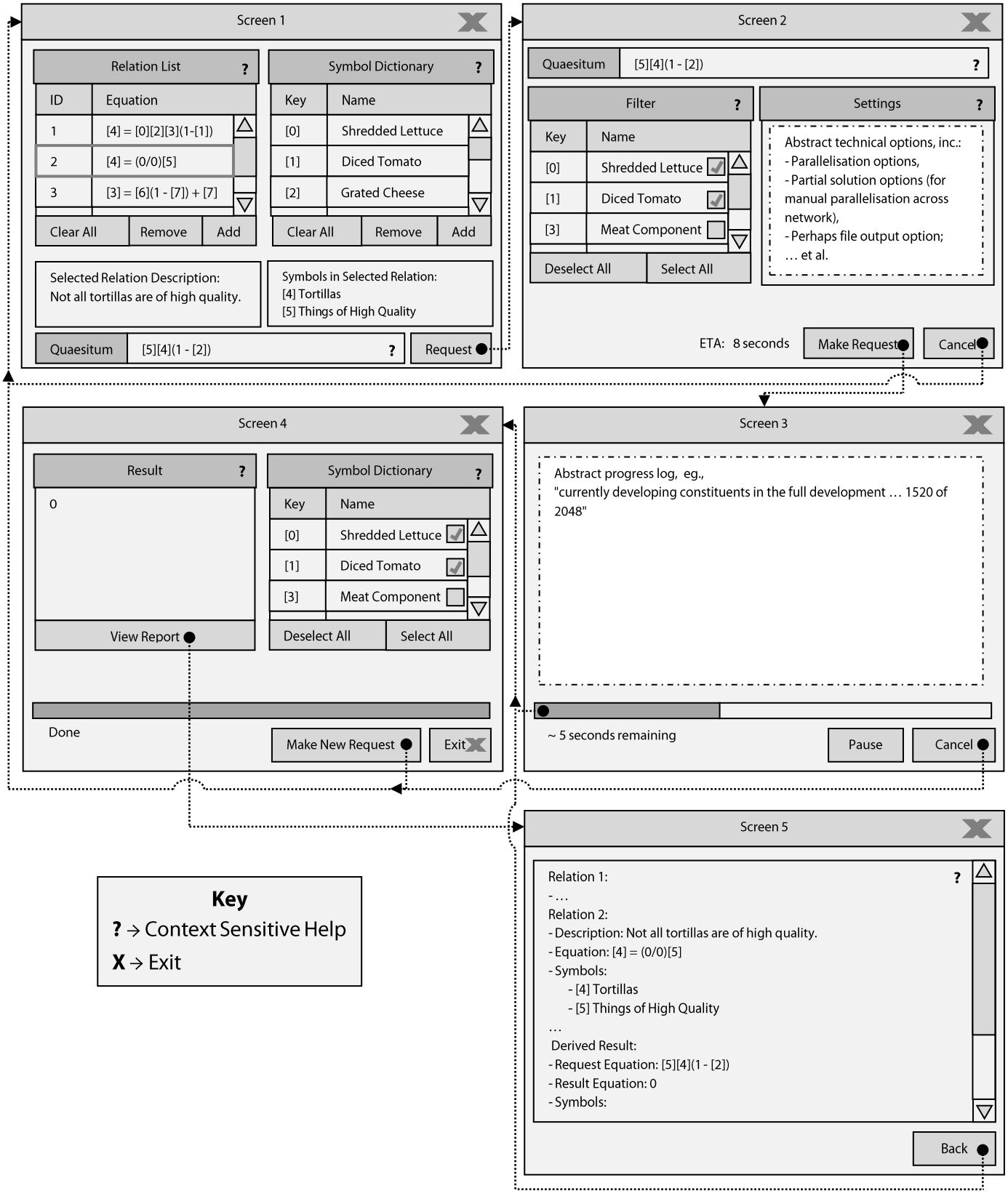
The systems flowchart warrants a remodel on the completion of the solution, but this primitive form is perfectly sufficient for development to continue.

(An appended note) -- a remodel does not find its place anywhere else in the report, so one that reflects the source appears on the next page. Still, the representation does not merely match the complexity of the system, but aims instead to express the important features. In reading any commentaries, it must be remembered that this has been added in post.


Symbol Details
Fact details
Validity of fact
Add Symbol
Symbol to Remove
Mathematical translation
Check Validity
Symbol List
Remove Symbol
Add Fact
Fact to Remove
Selected Fact
Remove Fact
Fact List
List Involved Symbols
Select Fact
List of Symbols in Selected Fact
Aggregate Information
Request
Request Fact
Develop Information with Respect to the Requested Fact
Adorn Result
Sought Information

2. USER INTERFACE

2.1. DRAFT GRAPHICAL USER INTERFACE



To this draft, features will iteratively be added and subtracted during the development process based on additional conference with end users and the client. A few additional observations can be made:

- The GUI is in a mostly sequential structure because the program itself is procedural.

- Progress feedback is provided while the P.L.E is doing the bulk of the evaluation.

- Links to help screens are present wherever necessary.

- Input validation is not described above for simplicity, but is of extreme importance.

- GUI artefacts are separated meaningfully.

3. PROGRAMMING PLAN

3.1. VARIABLE MANAGEMENT

The recognition of the need for individual abstract data structures is a step that must be done before a programming plan can be generated. The most appropriate data structures will be chosen on the basis of speed, memory, and either simplicity to implement correctly or existence in already written libraries. The preliminary data dictionary follows -- but there will be several iterations, and this one is certainly of very low detail. It contains as a minimum everything observable in the data flow diagram (`III. 1.2.) and the storyboard ('III. 2.1.).

In the data dictionary below,:

- "ADT" confers that a class is an abstract data types, and thus, some or all of its properties will be undefined, as denoted by an empty shaded row. The extrinsic properties are explored as appropriate in section `III. 3.2..

- The "INDETERMINATE" keyword is used in cells where information cannot be provided.

- The tilde '~' character surrounds some text in the Example column where an example is represented in a form which it does not necessarily appear in if unobserved. Tildes are used to express where the author has simplified or changed the form of an example for ease of interpretation.

- '_' proceeds a variable name to denote that that variable is an instance of a class, that is, _Meatball is a variable which is a generic instance of the class Meatball.

- [a, b, …] represents an array of elements 'a', 'b', … -- each of the same type.

- {a: _a, b: _b} represents a class with properties 'a' and 'b', respectively holding the values _a and _b.

- In the Formatting column, "Relation" warrants a reference to section `III. 1.2., wherein it is defined in EBNF.

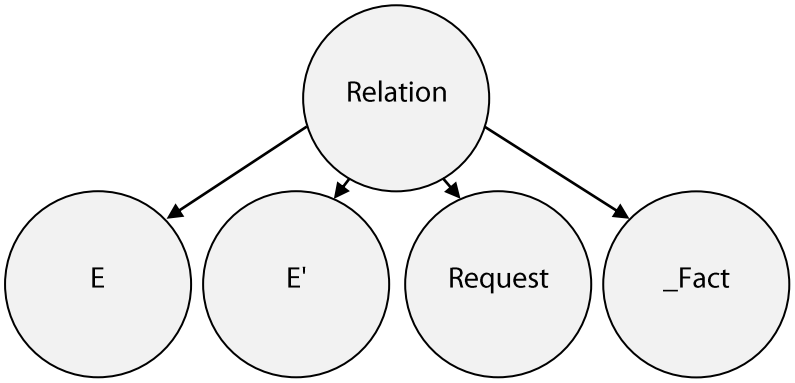
- Indented data items are fields of the last mentioned (sequentially) record.

Data Item	Type	Formatting	Size (Bytes)	Example
	Description			
_Fact	(ADT) Record	Relation	INDETERMINATE	~All men are mortal. ~
	Data type that encapsulates a qualitative relationship between logical symbols. Facts are provided directly by the user.			
.description	String	Unrestricted	O(n) ∝ string length	"All men are mortal."
	Stores a worded description of a fact that otherwise exists in only a mathematical form. The description serves no functional purpose other than to be shown to the user to aid understanding of equations.			

Data Item	Type	Formatting	Size (Bytes)	Example
	Description			
_Symbol	Record	{name: _name, hash: _hash}	O(n) ∝ string length of .name, (+ 4)	{name: "Octopuses", hash: 8}
	A symbol stores a "thing" upon which logical restrictions can be applied. Has the intrinsic property of a name, and extrinsically, is identified by a number.			
.hash	Integer	Positive: n ≥ 0. Unique.	4	12
	Stores the ID of the symbol. Each symbol must have a unique hash. The system uses hashes to reference symbols in mathematical equations, rather than referencing them by the longer property of name.			
.name	String	Unrestricted	O(n) ∝ string length	"Things with eight arms"
	Stores the name of a symbol so as to allow the user to identify it.			
developedRequest	(ADT) Record	INDETERMINATE	$O(2^n)$ ∝ array length of symbolsInRequest	INDETERMINATE
	An expanded form of "request". Contains information on what constituents have coefficients of 0 or 1 (disappear or do not vanish), thus being an intermediary in the determination of what coefficients do and do not vanish.			
.coefficientMap	Array of Booleans	[0 | 1, 0 | 1, …]	O(n) ∝ array length. The length happens to equal 2^n ∝ array length of symbolsInRequest	[0, 1, 0, 0, 0, 1, 0, 1]
	Confers the constituents that do and do not disappear. Each constituent is internally assigned a unique value (by some reversible algorithm, such that the constituent can be divined from its integral value), and the corresponding entry in this array is the coefficient.			
factAggregation	(ADT) Record	INDETERMINATE	$O(2^n)$ ∝ number of symbols developed with respect to.	~No men placed in exalted stations are free from envious regards. Henry is human. Henry is placed in an exalted station. ~
	Stores the total knowledge of the system. Upon which, transformations can be applied such that the factAggregation can be flagged as "prepared" or not.			
.isPrepared	Boolean	0 | 1	1 byte	1
	Flag for the state of the factAggregation: stores 1 if the structure has been developed with respect to the symbols in the request, and 0 otherwise.			

Data Item	Type	Formatting	Size (Bytes)	Example
	Description			
factList	Array of Facts	[Fact, Fact, …]	O(n) ∝ array length	[~All wealth is transferable~, ~wealth is limited in supply~]
	Stores the list of facts that equate to the sum of all knowledge within the system.			
filter	Array of Symbols (Array of Records)	[Symbol, Symbol, …]	O(n) ∝ array length	[{name: "Dogs", hash: 4}, {name: "Shiba Inus", hash: 5}]
	Holds the list of symbols that soughtRelation will reference. This array is used to adorn the result before it reaches the user.			
nonVanishingCoefficientsProduct Shorthand: E	(ADT) Record	INDETERMINATE	INDETERMINATE	INDETERMINATE
	Stores the mathematical product of all coefficients of the prepared factAggregation whose respective coefficients in developedRequest are equal to 1.			
request	(ADT) Record	Relation	INDETERMINATE	~Dogs that are playful~
	The user directly inputs the request, which will be used to solve for the sought information.			
soughtRelation	(ADT) Record	INDETERMINATE	$O(2^n)$ ∝ array length of filter	~Henry is regarded with envy, and is not an octopus.~
	The return result of the program. It can have a filter applied to it so as to reduce its size. It is the result of calculating an analogy to the algebraic expression E' / (E' - E), where E' vanishingCoefficientsProduct, and E = nonVanishingCoefficientsProduct.			
symbolsInRequest	Array of Symbols (Array of Records)	[Symbol, Symbol, …]	O(n) ∝ array length	[{name: "Times when it thunders", hash: 2}, {name: "Times when it rains", hash: 4}]
	A list of symbols that are in the request, with which factAggregtion is to be developed with respect to in order to become "prepared".			
vanishingCoefficientsProduct Shorthand: E'	(ADT) Record	INDETERMINATE	INDETERMINATE	INDETERMINATE
	Stores the mathematical product of all coefficients of the prepared factAggregation whose respective coefficients in developedRequest are in it for 0.			

In considering variables, their common properties cannot be ignored. Because of the problems mathematical roots (`III. 1.), it comes to easy apprehension that many of the abstract data types share a common template: the relation class.



Namely, E, E', Request, _Fact; are all perfectly well formed relations. It is warranted that, in order to investigate how data could feasibly stored within the program, a base class with such a multiplicity of children should be rigorously defined. This definition ought to be mathematical, rather than just syntactical as it has hereto been.

Knowledge is the affirmation or denial of existence of certain things -- things which are represented by combinations of symbols. Let us call this combination of symbols M, then knowledge can be represented as either:

M = 1 … (0);
or
M = 0 … (1);

From (0),
1 - M = 0;

But it is known that M and 1 - M -- abstractly -- refer to the same thing: they are both combinations of symbols. Thus relations can be generalised into the preferable form:

M = 0;

Now, M is symbolic of some indeterminate number of things, such that -- listing all things within the universe of discourse -- a coefficient of either 0 or 1 can be attached to make the expression valid. In other words, M is an expression such that, given a string of Boolean inputs, it returns a Boolean to answer whether or not the given inputs exist within the domain.

The base relation class M can be generalised from the IO properties given in the following table.

Inputs (Arguments)					Output
A_0	A_1	A_2	A_3	A_4	Coefficient
0	0	0	0	0	θ_0
0	0	0	0	1	θ_1
0	0	0	1	0	θ_2
0	0	0	1	1	θ_3
0	0	1	0	0	θ_4
0	0	1	0	1	θ_5
0	0	1	1	0	θ_6
0	0	1	1	1	θ_7
0	1	0	0	0	θ_8
0	1	0	0	1	θ_9
0	1	0	1	0	θ_{10}
0	1	0	1	1	θ_{11}
0	1	1	0	0	θ_{12}
0	1	1	0	1	θ_{13}
0	1	1	1	0	θ_{14}
0	1	1	1	1	θ_{15}
1	0	0	0	0	θ_{16}
1	0	0	0	1	θ_{17}
1	0	0	1	0	θ_{18}
1	0	0	1	1	θ_{19}
1	0	1	0	0	θ_{20}
1	0	1	0	1	θ_{21}
1	0	1	1	0	θ_{22}
1	0	1	1	1	θ_{23}
1	1	0	0	0	θ_{24}
1	1	0	0	1	θ_{25}
1	1	0	1	0	θ_{26}
1	1	0	1	1	θ_{27}
1	1	1	0	0	θ_{28}
1	1	1	0	1	θ_{29}
1	1	1	1	0	θ_{30}
1	1	1	1	1	θ_{31}

The table gives the extrinsic properties of a relation, but further mathematical foundation is required in many models.

Applying Lagrange's interpolative polynomial with binary inputs across 5 dimensions gives an instance of the mathematical expression M. Making observation of (or recursively applying across the binary domain)

$$f(n)=\sum_{i=0}^{d}\left\{y_i\left[\prod_{k=0}^{i-1}\left\{\frac{(n-x_k)}{(x_i-x_k)}\right\}\prod_{k=i+1}^{d}\left\{\frac{(n-x_k)}{(x_i-x_k)}\right\}\right]\right\},\ \frac{d^{d+1}f(n)}{dn^{d+1}}=0;$$

Yields the table that follows, whose pattern is easy to follow:

Constituents (Parameters)					Output
A_0	A_1	A_2	A_3	A_4	Coefficient
$(1 - A_0)$	$(1 - A_1)$	$(1 - A_2)$	$(1 - A_3)$	$(1 - A_4)$	θ_0
$(1 - A_0)$	$(1 - A_1)$	$(1 - A_2)$	$(1 - A_3)$	A_4	θ_1
$(1 - A_0)$	$(1 - A_1)$	$(1 - A_2)$	A_3	$(1 - A_4)$	θ_2
$(1 - A_0)$	$(1 - A_1)$	$(1 - A_2)$	A_3	A_4	θ_3
$(1 - A_0)$	$(1 - A_1)$	A_2	$(1 - A_3)$	$(1 - A_4)$	θ_4
$(1 - A_0)$	$(1 - A_1)$	A_2	$(1 - A_3)$	A_4	θ_5
$(1 - A_0)$	$(1 - A_1)$	A_2	A_3	$(1 - A_4)$	θ_6
$(1 - A_0)$	$(1 - A_1)$	A_2	A_3	A_4	θ_7
$(1 - A_0)$	A_1	$(1 - A_2)$	$(1 - A_3)$	$(1 - A_4)$	θ_8
$(1 - A_0)$	A_1	$(1 - A_2)$	$(1 - A_3)$	A_4	θ_9
$(1 - A_0)$	A_1	$(1 - A_2)$	A_3	$(1 - A_4)$	θ_{10}
$(1 - A_0)$	A_1	$(1 - A_2)$	A_3	A_4	θ_{11}
$(1 - A_0)$	A_1	A_2	$(1 - A_3)$	$(1 - A_4)$	θ_{12}
$(1 - A_0)$	A_1	A_2	$(1 - A_3)$	A_4	θ_{13}
$(1 - A_0)$	A_1	A_2	A_3	$(1 - A_4)$	θ_{14}
$(1 - A_0)$	A_1	A_2	A_3	A_4	θ_{15}
A_0	$(1 - A_1)$	$(1 - A_2)$	$(1 - A_3)$	$(1 - A_4)$	θ_{16}
A_0	$(1 - A_1)$	$(1 - A_2)$	$(1 - A_3)$	A_4	θ_{17}
A_0	$(1 - A_1)$	$(1 - A_2)$	A_3	$(1 - A_4)$	θ_{18}
A_0	$(1 - A_1)$	$(1 - A_2)$	A_3	A_4	θ_{19}
A_0	$(1 - A_1)$	A_2	$(1 - A_3)$	$(1 - A_4)$	θ_{20}
A_0	$(1 - A_1)$	A_2	$(1 - A_3)$	A_4	θ_{21}
A_0	$(1 - A_1)$	A_2	A_3	$(1 - A_4)$	θ_{22}
A_0	$(1 - A_1)$	A_2	A_3	A_4	θ_{23}
A_0	A_1	$(1 - A_2)$	$(1 - A_3)$	$(1 - A_4)$	θ_{24}
A_0	A_1	$(1 - A_2)$	$(1 - A_3)$	A_4	θ_{25}
A_0	A_1	$(1 - A_2)$	A_3	$(1 - A_4)$	θ_{26}
A_0	A_1	$(1 - A_2)$	A_3	A_4	θ_{27}
A_0	A_1	A_2	$(1 - A_3)$	$(1 - A_4)$	θ_{28}
A_0	A_1	A_2	$(1 - A_3)$	A_4	θ_{29}
A_0	A_1	A_2	A_3	$(1 - A_4)$	θ_{30}
A_0	A_1	A_2	A_3	A_4	θ_{31}

The sum of the product of each row is the mathematical expression sought. This is the information that each variable that is a subclass to "Relation" must include intrinsically (although perhaps in a varying but equivalent form), and the part of the relation "= 0" can be ignored since it is common to each different relation.

Conclusively, the development of the Relation class is of paramount importance to the modelling of the problem, and is deserving of particular attention. The Relation object type must be able to store an analog to the tables above, where a given set of inputs in a request is mapped to a stored Boolean value. All subclasses of the Relation superclass will be in a form analogous to the mathematical expression $0 = \delta$.

3.2. INTERFACE DESIGN

Having the data that is to be stored already defined, it remains that the manipulation of this data is planned. The interface will be the foundation of the GUI (`III. 2.1.), and will be more directly accessible from the CLI. Detail will develop naturally in concurrency with the programming of the core, but a set of standards must be made before the programming begins.

The development of function interfaces helps model the system into modular components which can be implemented individually with ease, and allows the preconstruction of test cases. This section is only a loose standard, and there will surely be modifications that render some of this work obsolete. Many helper functions will likely appear in the implementation which are not present here, and functions such as operator overloads have been omitted.

FactDatabase			
Return Type	Function Name	Parameter Type	Parameter Name
	appendFact	Relation	fact
	deleteFact	int	factNumber
	FactDatabase		
Relation	getFactAggregation		
Relation[]	listFacts		

Fraction			
Return Type	Function Name	Parameter Type	Parameter Name
	Fraction	Relation	numerator
		Relation	denominator
Relation	getDenominator		
Relation	getNumerator		

IOStream			
Return Type	Function Name	Parameter Type	Parameter Name
Relation	getFact		
int[]	getFilter		
Relation	getRequest		
	giveSoughtInformation	Fraction	soughtInformation
	IOStream		

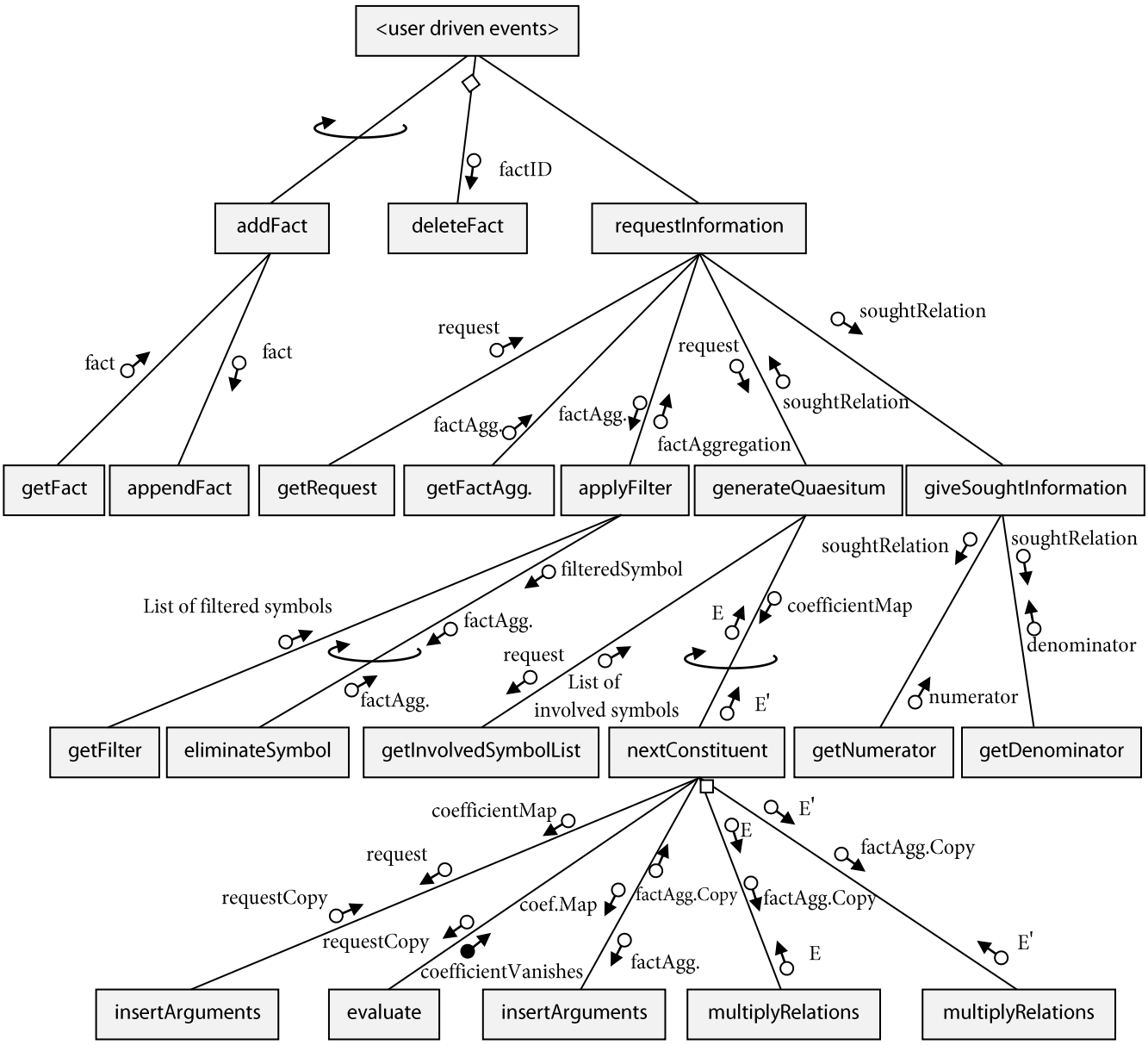
main			
Return Type	Function Name	Parameter Type	Parameter Name
Relation	addFact	FactDatabase	database
		Relation	fact
Relation	applyFilter	Relation	factAggregation
		int[]	symbolsToEliminate
	main		
Fraction	requestInformation	Relation	subject
		FactDatabase	database

Relation			
Return Type	Function Name	Parameter Type	Parameter Name
	eliminateSymbol	int	symbolHash
bool	evaluate		
int[]	getInvolvedSymbolList		
	insertArguments	triad[]	symbolValues
	Relation	string	constructor

By triad is meant a "char", taking the values 0, 1, or a representation of quantum indeterminacy (0/0).

3.3. MODULAR RELATIONSHIPS

Loosely based off of the function prototypes and data definitions, a preliminary structure chart can be created to aid in the modelling of the solution, correlating directly with less effort needed in the implementation stage. There have been some abbreviations and minor omissions for simplicity, and due to the independence of the GUI.



3.4. TEST PLAN

Due to the inherent complexity of results of the solution, and the lack of ease in interpreting results, testing will the most vital stage of the development life cycle. It is important, thus, that the paradigm of testing is developed beforehand. Testing the correctness of the solution will occur at several levels: deskchecking logic, unit testing implementation, and acceptance and live tests of the solution wholly.

Deskchecking will occur in the planning stage, and all levels of the solution will be tested -- not just what can be seen in 3.3. as being the end nodes. For each module, test cases will be planned in advance of implementations as an abstract algorithm. Once pseudocode has been developed, the algorithm will manually be stepped through with respect to the planned and perhaps additional test cases. All calls to subroutines will be assumed to return the correct result, and this assumption is verified by the testing of the individual subroutine. It is imperative in this stage that invalid inputs are not neglected from the testing scope.

Prior to the actual implementation of the program (yet during the implementation stage) test suites for each module will be developed. This is a form of black box testing: the unit tests will either pass or fail, and will not give insight into how the algorithm itself is fallible. Once a module is fully implemented, the tests will be written again in more detail. Most modules must pass its own set of unit tests to be classed as correct, but in this stage, the solution will not be tested as a whole program, and the more parental modules may go untested.

Acceptance testing occurs following the development of proper interfaces. The program is tested wholly externally through the CLI and GUI, and will be provided to a representative portion of the end users for review in beta testing so as to ensure that personnel can easily operate the solution. Task analysis will be performed so that the software is in line with how the common user is inclined to operate it. Live stress tests will occur in attempt to expose a multitude of errors, namely, those instigated by the concurrent processing of a high magnitude of inputs. Further, invalid data will intentionally be inputted so as to test how the program parses and sanitises inputs, thus asserting robustness. The program will be evaluated against the specification, which will reveal where modifications must be made so that the solution does exactly what it was intended to do.

Once the logic of the solution has been tested in these three phases, it gains the quality criteria of being correct, accurate, and reliable independent of external factors. What must next be tested is interoperability in terms of hardware and other software. Beta testing will have somewhat informally tested the software's ability to run with different hardware resources and software configurations, but this must explicitly be tested further.

FactDatabase::appendFact		
Abstract Object State	Input	Expected Output State
[]	fact1	[fact1]
[fact1]	fact2	[fact1, fact2]
[fact1, fact2]	fact3	[fact1, fact2,fact3]
[fact1]	"pulled pork"	Error: "pulled pork" is not of type Relation.

It can be noticed that "fact1", "fact2" &c are generic facts whose properties are unspecified. It is answered in justification that the intrinsic properties of these Relation classes are never observed by the appendFact function, and thus, they can remain abstract in the testing process. This module is not concerned with what exactly a fact stores, rather, it notices facts from an entirely external standpoint, where the only necessary properties are that they exist and can be stored. A fact storing "0" will behave exactly the same as a fact storing "1" within the scope of this function.

This appendFact function is one of a few that can easily be proven correct without any conventional walkthrough or tests, and this may be how procedure is conducted if it is seen as auspicious (particularly in terms of time). The Expected Output State of the 0^{th} case can be used as an invariant for the input of the 1^{st} case, and then recursively (by mathematical induction), each case can be proven provided the first two work.

FactDatabase::deleteFact		
Abstract Object State	Input	Expected Output State
[]	-1	Error: Input must be ≥ 0.
[]	0	Error: Fact does not exist.
[fact1]	1	Error: Fact does not exist.
[fact1]	0	[]
[fact1, fact2]	0	[fact2]
[fact1, fact2]	1	[fact1]
[fact1]	"chopped capsicum"	Error: "chopped capsicum" is not of type int.

Throughout the planning, it has been assumed that arrays carry their own length property. Although this is the case for many languages, there is a sizable portion where length must explicitly be declared (especially languages in the neighbourhood of assembly, which are concerned specifically with memory). If a language is chosen for implementation, there are two alternatives to correct this simplification -- the first being add length parameters and properties when necessary, and the second being to define a container class which stores both an array and a length property.

FactDatabase::FactDatabase

This constructor doesn't do anything that will be testable until the implementation stage (if at all). This function is called whenever a new FactDatabase is created, and it has been omitted from the structure chart in `3.3. due to its triviality.

FactDatabase::getFactAggregation	
Abstract Object State	Expected Output
[]	0
[fact1]	fact1
[fact1, fact2]	fact1 + fact2 · (1 - fact1)
[fact1, fact2, fact3]	fact1 + fact2 · (1 - fact1) + fact3 · (1 - fact2) · (1 - fact1)

There can be noticed similitude in both this function and appendFact in what was said earlier; both functions treat facts as abstract. It is a lot more difficult to get away with here, yet still, each fact can take any form and the function ought to work in the exact same way, even though they have operations such as addition transacted upon them.

FactDatabase::listFacts	
Abstract Object State	Expected Output
[]	[]
[fact1]	[fact1]
[fact1, fact2]	[fact1, fact2]
[fact1, fact2, fact3]	[fact1, fact2,fact3]

Fraction::Fraction			
Abstract Object State	Input1 (param: numerator)	Input2 (param: denominator)	Expected Output State
{}	relation1	relation2	{numerator: relation1, denominator: relation2}

This constructor simply stores the data that it is given as arguments, and its correctness will be intuitively discernible.

Fraction::getDenominator	
Abstract Object State	Expected Output
{numerator: relation1, denominator: relation2}	relation2

This function (and getNumerator) act as one-to-one interfaces between a Fraction and the outside world, and in most possible implementations of the Fraction class, they do very little. A time can be conceived of where the internal structure of Fraction confounds the process for other benefits, but such a case would probably warrant an external rewriting of the Fraction class rather than conforming to arbitrary theoretical interfaces.

Fraction::getNumerator	
Abstract Object State	Expected Output
{numerator: relation1, denominator: relation2}	relation1

IOStream::getFact	
Abstract User Input	Expected Output
"<subject> = <predicate>"	"<subject> · (1 - <predicate>) + <predicate> · (1 - <subject>)"
"guacomole"	Error: "guacomole" does not conform to Boolean restrictions.

The contents of <subject> and <predicate> do not perturb this function's operations, but it should be brought to attention that the input to this function isn't provided through a parameter. This function interfaces directly with the user (or a spoof operator) to collect the input. The IOSteam class exists to allow extensibility, interoperability, and usability -- it acts as a very small scale virtual machine, in happenstance analogy to the glorious JVM. If a custom user interface is to be built, the builder will

create their own implementation of the IOStream object while conforming to the predefined interfaces that have been drafted in `3.2.. The user input can take any form that the writer is able to correctly implement, and "<subject> = <predicate>" is just an example. The same applies for the next function getFilter. These functions will not necessarily warrant implementation in the algorithm planning stage (part`4.).

IOStream::getFilter	
Abstract User Input	Expected Output
is0Highlighted = 0, is1Highlighted = 0, is2Highlighted = 0, is3Highlighted = 0	[]
is0Highlighted = 1, is1Highlighted = 1, is2Highlighted = 1, is3Highlighted = 1	[0, 1, 2, 3]
is0Highlighted = 0, is1Highlighted = 1, is2Highlighted = 0, is3Highlighted = 1	[1, 3]
is0Highlighted = 1	[1]

IOStream::getRequest	
Abstract User Input	Expected Output
"<subject> = <predicate>"	<subject> · (1 - <predicate>) + <predicate> · (1 - <subject>)
"ground beef"	Error: "ground beef" does not conform to Boolean restrictions.

This function, as it appears here, is functionally congruent to getFact. The only reason for the division is that they both occur in different context, and it would be terrible naming practice to agglomerate them into one function; even though they both request things of the same type, there is a very different purpose behind each.

IOStream::giveSoughtInformation	
Abstract Object State	Abstract Expected Output (Printed To Screen)
{numerator: "[1]", denominator: "[1]"}	"[1] + 0/0(1 - [1])"
{numerator: "[2]", denominator: "[0]"}	"[0][2] + 0/0(1 - [0])(1 - [2])"
{numerator: "1 - [1]", denominator: "[0][1]"}	"0/0(1 - [0])[1]"
{numerator: "[1][0] + (1 - [0])", denominator: "1"}	"[1][0] + (1 - [0])"
{numerator: "1 - [2][7]", denominator: "0"}	"0/0[2][7]"
{numerator: "0", denominator: "0"}	"0/0"
{numerator: "1", denominator: "0"}	"0"
{numerator: "0", denominator: "1"}	"0"
{numerator: "1", denominator: "1"}	"1"

This function is the first (in the alphabetical sequence) to operate on the intrinsic properties of a Relation class, and it is also the first where finer functionality of the program must be tested. Hereto, it has easily been readily verifiable whether the expected output actually matches what the input warrants, but here, the friendly intermediary mathematics is required. Due to the inherent complexity, the tests are designed to reach into some of the deeper corners of functionality that is required of this module. The last four tests are designed specifically to exhaustively test the possible types of outputs, and the prior tests consider how these individual parts are combined.

This case (although there have been others) exemplifies the need for some ability that must be given to the tester to determine whether two Relations are equivalent For example, "1 - [0][1]"and "[0](1 - [1]) + (1 - [0])" are equal expressions, although this is not obvious. During deskchecking, parity can be manually verified, or CASE tools such as JavaScript snippets can be used to tell whether two Relations are equal, but this solution becomes impossible when unit tests are implemented. Thus, eventually, the "==" operator for the Relation class must be overloaded (or equivalent).

IOStream::IOStream

main::addFact		
Database Input State (param: database)	Input (param: fact)	Expected Database Output
[]	fact1	[fact1]
[fact1]	fact2	[fact1, fact2]
[fact1, fact2]	fact3	[fact1, fact2, fact3]
[fact1, fact2, fact3]	"sour cream"	Error: "sour cream" is not of type Relation.

This function acts the same as appendFact method, except it is done so externally. At this moment, this function has become redundant, and should really be encapsulated within IOStream::getFact. In the structure chart (part `3.3), it can be noticed that this function makes a call to the IOStream rather than accepting its Relation via a parameter, which is much less nonsensical.

What was perhaps wrong is left as is, for to attempt to remodel the interfaces outside of the actual algorithm design requires the informal conjuring of an algorithm design without ever instantiating it on paper, and the design will surely be of less quality thus. This decision may however render these exact tests redundant, but it is essential to recognise that the structure chart will differ from the steps in the IPO diagram, which will differ from the draft interface design and tests, which will differ from the pseudocode implementation, which will be discrepant from the actual implementation. The software design process is iterative, even when a wholly structured approach is observed.

main::applyFilter		
Abstract Object State (param: factAggregation)	Input (param: symbolsToEliminate)	Expected Output
"(1 - [3])[0][1] + (1 - [3])[2](1 - [0][1]) + [3]"	[]	"(1 - [3])[0][1] + (1 - [3])[2](1 - [0][1]) + [3]"
"(1 - [3])[0][1] + (1 - [3])[2](1 - [0][1]) + [3]"	[4]	"(1 - [3])[0][1] + (1 - [3])[2](1 - [0][1]) + [3]"
"(1 - [3])[0][1] + (1 - [3])[2](1 - [0][1]) + [3]"	[3]	"[0][1] + [2](1 - [0][1])"
"(1 - [3])[0][1] + (1 - [3])[2](1 - [0][1]) + [3]"	[2, 3]	"[0][1]"
"[0][1] + [2](1 - [0][1])"	[1]	"[2]"
"[2]"	[0, 1]	"[2]"
"[11]"	[1, 11]	"0"
"[2][1]"	[1]	"0"
"[2][0]"	[0, 1, 2]	"0"

"[1][2][3] + [1](1 - [2])(1 - [3]) + (1 - [1])(1 - [2])"	[2]	"0"
"[1][2][3] + [1](1 - [2])(1 - [3]) + (1 - [1])(1 - [2])"	[1, 2]	"0"
"[10][2][3] + [10](1 - [2])(1 - [3]) + (1 - [10])(1 - [2])"	[10]	"(1 - [2])(1 - [3])"
"1"	["shredded chicken"]	Error: "shredded chicken" is not of type integer.

Although the tests look vast here, the work is really delegated to Relation::eliminateSymbol, and the tests will be quite short; each function assumes the correctness of all other functions, thus simplifying all testing greatly.

In the test cases above, the term [10] sticks out quite a bit, and this is by test design. The inclusion of this as opposed to [9] is strategically inserted so that string handling is properly tested: hereto, all Relation symbols have taken the form [<digit>], and this is a particularly dangerous trap with the string type, for example, 9 and 10 are treated quite similarly as integers, but "9" and "10" are incomprehensibly different strings. [10] tests the case where a symbol adheres to the [<digit> <digit>] structure. The correctness may be able to be extrapolated to a sequence of three or more digits, but internal and dedicated volume tests should be ascribed this burden.

main::main

This is the main method. Although it ties everything together and is extremely important therefore, it is quite hard to test when a proper user interface has not yet been developed. The purpose of deskchecking in this development is primarily to assert the correctness of modular components. Live / acceptance tests will be dedicated to this function and its most significant child requestInformation, which follows.

main::requestInformation		
Abstract Object State (param: database)	Input (param: subject)	Expected Output
["[0]"]	[0]	{numerator: "0", denominator: "1"}
["[0](1 - [1]) + [1](1 - [0])"]	[0]	{numerator: "[1]", denominator: "1"}
["[0](1 - [2])", "[2]"]	"diced tomato"	Error: "diced tomato" is not of type Relation.

Relation::eliminateSymbol		
Abstract Object State	Input	Expected Output
"(1 - [3])[0][1] + (1 - [3])[2](1 - [0][1]) + [3]"	4	"(1 - [3])[0][1] + (1 - [3])[2](1 - [0][1]) + [3]"
"(1 - [3])[0][1] + (1 - [3])[2](1 - [0][1]) + [3]"	3	"[0][1] + [2](1 - [0][1])"
"[0][1] + [2](1 - [0][1])"	2	"[0][1]"
"[0][13] + [2](1 - [0][13])"	2	"[0][13]"
"1"	0	"1"
"[14]"	14	"0"

"[10][2][3] + [10](1 - [2])(1 - [3]) + (1 - [10])(1 - [2])"	10	"(1 - [2])(1 - [3])"
"1 - [15]"	"melted cheese"	Error: "melted cheese" is not of type integer.
"[2] + [1](1 - [2])"	-417	Error: Input must be ≥ 0.

Relation::evaluate	
Abstract Object State	Expected Output
"(0)"	0
"(1)"	1
"1"	1
"(1)(1 - (0))"	1
"(0)(1)"	0
"(0)(1 - (0)) + (0)(1 - (1)) + (1) (1 - (0))(1 - (1 - (1)))"	1
"(1)(1)(0) + (1)(1 - (1))(1 - (0)) + (1 - (1))(1 - (1))"	0
"(1 - (1))(0)(1) + (1 - (1))(0)(1 - (0)(1)) + (1)"	1
"(1)(1)(0) + (1)(1 - (1))(1 - (0)) + (1 - (1))(1 - (1))"	0
"[0]"	Error: not all symbols eliminated

This function is of quite a strange nature, and may not see an implementation in pseudocode. Many languages come with an eval function which includes the functionality that is here required, but at quite a cost (in terms of speed see `III. 5., as well as carrying a multitude of other detriments). For example, in defining these test cases, I did not manually evaluate LHS, I simply used mathematical faculties provided by Google to evaluate them, and this makes it sufficiently obvious that this module has been implemented many times, and is viable for reuse. It is unfortunate that in much of the rest of the solution, custom logic is required, and prebuilt libraries or functions cannot be used without heavy modification.

Even if a prebuilt eval function is included (whether being just for mathematical expressions of for code), Relation::evaluate should act as a middle-man so that changes can be made in the future without having to do major refactoring (although appropriate CASE tools trivialise refactoring).

Relation::getInvolvedSymbolList	
Abstract Object State	Expected Output
"[10]"	[10]
"0"	[]
"[0] + [1](1 - [0])"	[0, 1]
"[0][1] + [2](1 - [0][1])"	[0, 1, 2]
"[2](1 - [4])"	[2, 4]
"[0] + (1 - [0])"	[]
"[1][0] + [1](1 - [0])"	[1]
"(0)[1]"	[]
"[12](1 - [12])	[]
"[4] + (1 - (0))(1 - [4])"	[]
"[4][0][1] + (1 - [0])[1](1 - [4]) + (1 - [0])[1][4] + (1 - [4])[0][1] + [0][4](1 - [1]) + [0](1 - [1])(1 - [4]) + (1 - [0])(1 - [1])"	[]

This function is quite easily shoddily implemented, so the exhaustive tests that can be seen are exalted in importance. Looking at the test case, it is hard not to think that the symbols [0], [1], and [4] must mean something, and it is unverifiable without paper that they do in fact collapse into unity.

Looking at the last case, I -- as a fallible human -- can hardly believe that I am not mistaken in the result I have written. Hereto, I haven't actually tested exhaustively that it is correct, I merely derived it from unity with garden-variety algebra, and I am wont to make algebraic errors. What then if I am wrong? Will the program conform to the incorrect test case, will I be perpetually confused as to why the expected and actual outputs don't match? Surely, in honesty, many of these test cases will be erroneous, and all that can be done is hope that the errors are of type 1 (in that they are false hits). Whenever a test hits, I will be sure to verify manually whether the test case is or is not wrong -- it's a kind of scepticism and distrust of one's past self that a programmer must develop to be successful; to be so inexorably confident in your own fallibility.

Relation::insertArguments		
Abstract Object State	Input	Expected Output State
"[4][0][1] + (1 - [0])[1](1 - [4]) "	['v', 'v', 'v', 'v', '1', 'v', '1']	"(1)[0][1] + (1 - [0])[1](1 - (1))"
"(1)[0][1] + (1 - [0])[1](1 - (1))"	[]	"(1)[0][1] + (1 - [0])[1](1 - (1))"
"(1)[0][1] + (1 - [0])[1](1 - (1))"	['v', 'v', '0']	"(1)[0][1] + (1 - [0])[1](1 - (1))"
"(1)[0][1] + (1 - [0])[1](1 - (1))"	['1']	"(1)(1)[1] + (1 - (1))[1](1 - (1))"
"[4]"	['v', 'v', 'v', 'v', '0', 'v', 'v', 'v', '1', '0']	"(0)"
"[0][1][2][3][4][5](1 - [10][12])"	['v', '1', '0', 'v', '1', '1', 'v', '1']	"[0](1)(0)[3](1)(1)(1 - [10][12])"
"[0]"	['v']	"[0]"
"[0](1 - [0]) + [0] + (1 - [0])"	['1']	"(1)(1 - (1)) + (1) + (1 - (1))"

Above, 'v' is the character that represents indeterminacy (0/0). This will not necessarily stay the case, and perhaps, in the future, any non '0' or '1' character will be treated as indeterminate. Because of the fact that this notation has not yet been decided, and that the "triad" type remains entirely theoretical, the test cases chosen are to be limited in variety.

Relation::Relation		
Abstract Object State	Input	Expected Output State
{}	<relation>	{data: <relation>}

4. ALGORITHM DESIGN

4.0. NOTES

Surely, the pseudocode in this section is erroneous, and it certainly delegates unwritten logic to functions that are not realised. In this scenario, due to the scope of the project, the algorithm is planned merely to ease the implementation; pseudocode certainly does not stand on its own. The division of purpose amongst domain of pseudocode is twofold: firstly, to allow possible designs to be explored, allow logic to be tested, and for analysis, and secondly, to explicitly document the logic. To both these ends, mild errors (mostly in the form of spelling errors or some short-sight) are not nearly disastrous.

There are no standards for pseudocode, but there is the imposition of the course specification document (which the reader must necessarily be familiar with). By liberty, some additions and modifications have been made so as to precipitate expressiveness:

- Array indexes are accessed using square-brackets rather than parentheses so as to avoid any ambiguity between function calls and memory access.

- Arrays are zero based (by habit of the developer, and convenience to the problem), meaning that the nth element of an array is accessed array[n - 1].

- The keyword "this" refers to the object whose method is currently being called --:

```
BEGIN Class.method(argument)
   this.otherMethod(argument)
END Class.method
```

In the snippet above, if object.method(argument) is called, then the first line of the method will enact the calling of object.otherMethod(argument).

- For loops have been simplified (I am of the opinion that this is tacit in the course specification document):

```
FOR incrementer = floor TO roof STEP 1
    // do something
NEXT incrementer
```

Is simplified to:

```
FOR incrementer = floor TO roof
    // do something
NEXT incrementer
```

- For loop boundaries work such that the following loop executes n times:

```
FOR incrementer = base TO base + n
    // do something
NEXT incrementer
```

- The "new" keyword makes a call to an objects constructor:

```
objectName = new Class(argument)
```

Is shorthand for:

```
Declare objectName as type Class
objectName.Class(argument)
```

- There exists one constant global variable, as here declared:

```
UNITY = new Relation("1") -- immutable
```

- "//" is indicative of the beginning of a comment line.

It must also be recognised that some features (one particularly) are omitted from the pseudocode. Most noticeably, the elimination of indeterminate values seems to escape mention, so with the design thus far, the user cannot query or push facts which involve any uncertainty. The logic behind this, however, is trivial -- to eliminate a symbol θ from δ, the result will be the product of the evaluation of δ with the parameter θ respectively set to naught and unity. Most practically, this would be implemented as a method that is called on the construction of a Relation.

4.1. VOID FACT_DATABASE::APPEND_FACT(RELATION FACT)

```
BEGIN FactDatabase.appendFact(fact)
    Append fact to this.factList
END appendFact
```

4.2. VOID FACT_DATABASE::DELETE_FACT(INT FACT_NUMBER)

```
BEGIN FactDatabase.deleteFact(factNumber)
    oldFactList = this.factList
    Set numFacts to number of elements in oldFactList
    IF factNumber < 0 OR factNumber >= numFacts THEN
        Display "Error, fact ", factNumber, " not found"
    ELSE
        Delete contents of this.factList
        FOR fact = 0 TO numFacts - 1
            IF (fact < factNumber) THEN
                sourceIndex = fact
            ELSE
                sourceIndex = fact + 1
            ENDIF
            this.appendFact(oldFactList[sourceIndex])
        NEXT fact
    ENDIF
END deleteFact
```

4.3. FACT_DATABASE::FACT_DATABASE(VOID)

```
BEGIN FactDatabase.factDatabase
    Initialise this.factList to an empty array
END factDatabase
```

4.4. RELATION FACT_DATABASE::GET_FACT_AGGREGATION(VOID)

```
BEGIN FactDatabase.getFactAggregation
    Set numFacts to number of elements in this.factList
    aggregate = new Relation("0")
    FOR fact = 0 TO numFacts
        productOfPriorFacts = new Relation("1")
        FOR encounteredFact = 0 to fact
            productOfPriorFacts *= UNITY - this.factList[encounteredFact]
        NEXT encounteredFact
        currentFact = this.factList[fact]
        aggregate += currentFact * productOfPriorFacts
    NEXT fact
    RETURN aggregate
END getFactAggregation
```

4.5. RELATION[] FACT_DATABASE::LIST_FACTS(VOID)

```
BEGIN FactDatabase.listFacts
    RETURN this.factList
END listFacts
```

4.6. FRACTION::FRACTION(RELATION NUMERATOR, RELATION DENOMINATOR)

```
BEGIN Fraction.Fraction(numerator, denominator)
    this.numerator = numerator
    this.denominator = denominator
END Fraction
```

4.7. RELATION FRACTION::GET_DENOMINATOR(VOID)

```
BEGIN Fraction.getDenominator
    RETURN this.denominator
END getDenominator
```

4.8. RELATION FRACTION::GET_NUMERATOR(VOID)

```
BEGIN Fraction.getNumerator
    RETURN this.numerator
END getNumerator
```

4.9. RELATION IO_STREAM::GET_FACT(VOID)

```
BEGIN IOStream.getFact
    Get factConstructor as string input from user
    // the input must be validated, but this is neglected in this
    // preliminary design --
    // TODO: validate
    Set stringLength to number of characters in factConstructor
    LHS = ""
    RHS = ""
    hasEncounteredCopula = false
    FOR character = 0 TO stringLength
        currentCharacter = factConstructor[character]
        IF currentCharacter == '=' THEN
            hasEncounteredCopula = true
        ELSEIF currentCharacter != ' ' THEN
            IF hasEncounteredCopula THEN
                RHS += currentCharacter
            ELSE
                LHS += currentCharacter
            ENDIF
        ENDIF
    NEXT character
    fact = new Relation(0)
    subject = new Relation(LHS)
    predicate = new Relation(RHS)
    fact += subject * (UNITY - predicate)
    fact += predicate * (UNITY - subject)
    RETURN fact
END getFact
```

This module is a partial stub implementation -- the IOStream interface is abstract, and can conceivably be many implementations. This particular implementation simply assumes that the user feeds the system a fact in the standard "<subject> = <predicate>" form, and avoids the task of having to validate input thence.

It should be noted that this is perhaps the only subroutine that has been optimised in any way, simply because it is implemented in such a way that it removes whitespace from a string, thus improving the efficiency of traversal and storage. This was done only because it circumstances were extraordinarily auspicious to such an improvement, and because it does not cause suffer in the universe of readability.

4.10. INT[] IO_STREAM::GET_FILTER(VOID)

```
BEGIN IOStream.getFilter
    Read isHighlighted from screen, containing a Boolean map of which
        symbols are highlighted
    Set numSymbols to number elements in isHighlighted
    Initialise highlightedSymbols to an empty array
    FOR symbol = 0 TO numSymbols
        IF isHighlighted[symbol] THEN
            Append symbol to highlightedSymbols
        ENDIF
    NEXT symbol
    RETURN highlightedSymbols
END getFilter
```

This function will take different forms depending on the type of input, that is, whether the program is running in console mode, with the standard GUI skin, other skins, mechanical interfaces, or virtual environments. As in the previous module (and in a way that is extensible to other IOStream methods), this is a dummy implementation.

4.11. RELATION IO_STREAM::GET_REQUEST(VOID)

```
BEGIN IOStream.getRequest
    Get requestConstructor as string input from user
    // TODO: validate
    Set stringLength to number of characters in requestConstructor
    LHS = ""
    RHS = ""
    hasEncounteredCopula = false
    FOR character = 0 TO stringLength
        currentCharacter = requestConstructor[character]
        IF currentCharacter == '=' THEN
            hasEncounteredCopula = true
        ELSEIF currentCharacter != ' ' THEN
            IF hasEncounteredCopula THEN
                RHS += currentCharacter
            ELSE
                LHS += currentCharacter
            ENDIF
        ENDIF
    NEXT character
    request = new Relation(0)
    subject = new Relation(LHS)
    predicate = new Relation(RHS)
    request += subject * (UNITY - predicate)
    request += predicate * (UNITY - subject)
    RETURN request
END getRequest
```

Code is reused, and in further iterations, it would irrefutably be beneficial to generate a delegate subroutine which treats facts and requests indiscriminately as relations. Such a function could be:

```
BEGIN transpose(relationConstructor)
    Set stringLength to number of characters in relationConstructor
    LHS = ""
    RHS = ""
    hasEncounteredCopula = false
    FOR character = 0 TO stringLength
        currentCharacter = relationConstructor[character]
        IF currentCharacter == '=' THEN
            hasEncounteredCopula = true
        ELSEIF currentCharacter != ' ' THEN
            IF hasEncounteredCopula THEN
                RHS += currentCharacter
            ELSE
                LHS += currentCharacter
            ENDIF
        ENDIF
    NEXT character
    relation = new Relation(0)
    subject = new Relation(LHS)
    predicate = new Relation(RHS)
    relation += subject * (UNITY - predicate)
    relation += predicate * (UNITY - subject)
    RETURN relation
END transpose
```

With the common module above, getRequest and getFact could be written as:

```
BEGIN IOStream.getFact
    Get factConstructor as string input from user
    // TODO: validate
    fact = transpose(factConstructor)
    RETURN fact
END getFact

BEGIN IOStream.getRequest
    Get requestConstructor as string input from user
    // TODO: validate
    request = transpose(requestConstructor)
    RETURN request
END getRequest
```

-- Conversely, this is an early iteration of the logic of the program, so the refinement of logic can be left to when experiments are actuated in the implementation stage.

4.12. VOID IO_STREAM::GIVE_SOUGHT_INFORMATION(FRACTION SOUGHT_INFORMATION)

```
BEGIN IOStream.giveSoughtInformation(soughtInformation)
    numerator = soughtInformation.getNumerator()
    denominator = soughtInformation.getDenominator()
    symbolList = getCombinedSymbolList(numerator, denominator)
    Set numSymbols to number of elements in symbolList
    numConstituents = 2 ^ numSymbols
    coefficientMap = new CoefficientMap(symbolList)

    // print every non-vanishing constituent to the screen
    FOR constituent = 0 TO numConstituents
        Initialise this.factList to an empty array
        Copy numerator into numeratorCopy
        Copy denominator into denominatorCopy
        numeratorCopy.insertArguments(coefficientMap.read())
        numeratorValue = numeratorCopy.evaluate()
        denominatorCopy.insertArguments(coefficientMap.read())
        denominatorValue = denominatorCopy.evaluate()

        IF numeratorValue == denominatorValue
            displayableConstituent = ""
            FOR symbol = 0 TO numSymbols
                IF coefficientMap.read()[symbol] == 0 THEN
                    displayableConstituent += "[", symbol, "]"
                ELSEIF coefficientMap.read()[symbol] == 1 THEN
                    displayableConstituent += "(1 - [", symbol, "])"
                ENDIF
            NEXT symbol
            IF numeratorValue == 1 THEN
                IF displayableConstituent == "" THEN
                    Display "1"
                ELSE
                    Display displayableConstituent
                ENDIF
            ELSE
                Display "0/0", displayableConstituent
            ENDIF
        ENDIF
        coefficientMap.increment()
    NEXT constituent
    IF nothing has been printed to the screen THEN
        Display "0"
    ENDIF

END giveSoughtInformation
```

```
BEGIN getCombinedSymbolList(relationA, relationB)
    involvedSymbols = relationA.getInvolvedSymbolList()
    Set numSymbolsInA to number of elements in involvedSymbols
    symbolsInB = relationB.getInvolvedSymbolList()
    Set numSymbolsInB to number of elements in symbolsInB
    FOR symbolInB = 0 TO numSymbolsInB
        currentSymbol = symbolsInB[symbolInB]
        collisionEncountered = false
        FOR symbolInA = 0 TO numSymbolsInA
            IF currentSymbol == involvedSymbols[symbolInA] THEN
                collisionEncountered = true
            ENDIF
        NEXT symbolInA
        IF NOT collisionEncountered THEN
            Append currentSymbol to involvedSymbols
        ENDIF
    NEXT symbolInB
    RETURN involvedSymbols
END getCombinedSymbolList
```

In future iterations, this method will likely be subjected to many improvements and optimisations, primarily, so that it outputs the resultant string of facts in a much more readable form, and whence, the quality criteria of usability can be more sufficiently met. As is, this function perhaps may print the more pithy result "All men are mortals" in the prolix version of the same thing: "all tall men are mortals, and all non-tall men are mortals", that is to say, the system does not recognise the fundamental reduction:

$\delta_1(1 - \delta_1) = 0$

$\delta_1^2 = \delta_1$

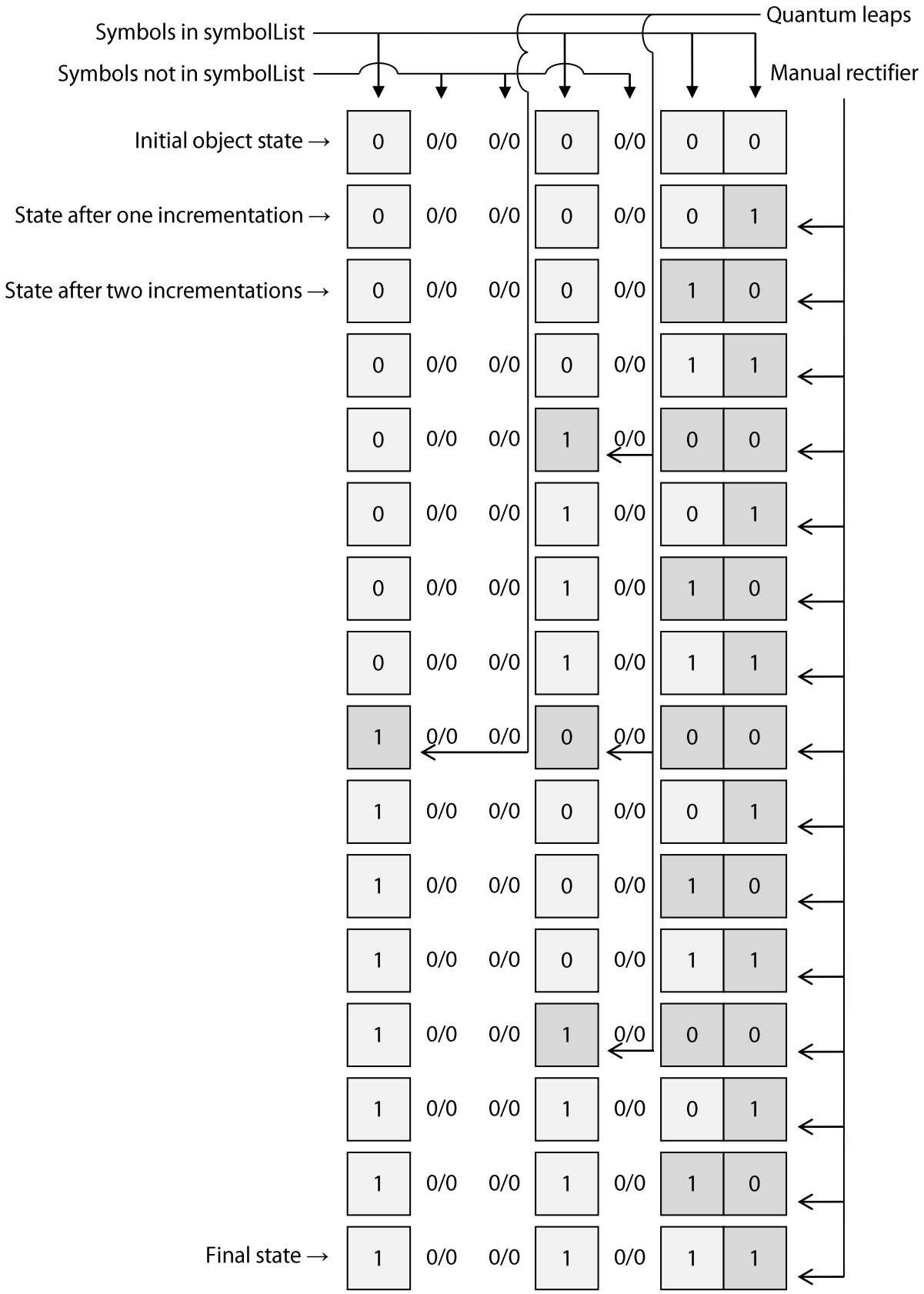
$\delta_1(1) = \delta_1$

$\delta_1(1 - \delta_2 + \delta_2) = \delta_1$

$\therefore \delta_1(1 - \delta_2) + \delta_1\delta_2 = \delta_1$

The program is still logically functional even when it outputs in such a form, so the adorning of this result is a concern that can be postponed for now.

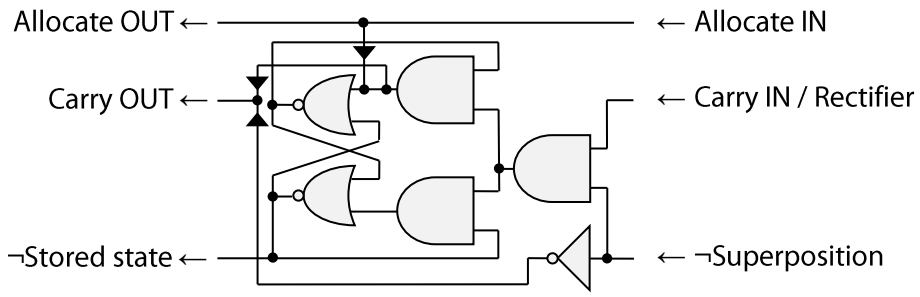
For simplicity, this subroutine supposes that there exists such a thing as a CoefficientMap class (dependencies are shaded in the pseudocode). The design of such a class can be best represented mechanically:


Quantum leaps
Symbols in symbolList
Symbols not in symbolList
Manual rectifier
Initial object state →
State after one incrementation →
State after two incrementations →
Final state →
0 0/0 0/0 0 0/0 0 0
0 0/0 0/0 0 0/0 0 1
0 0/0 0/0 0 0/0 1 0
0 0/0 0/0 0 0/0 1 1
0 0/0 0/0 1 0/0 0 0
0 0/0 0/0 1 0/0 0 1
0 0/0 0/0 1 0/0 1 0
0 0/0 0/0 1 0/0 1 1
1 0/0 0/0 0 0/0 0 0
1 0/0 0/0 0 0/0 0 1
1 0/0 0/0 0 0/0 1 0
1 0/0 0/0 0 0/0 1 1
1 0/0 0/0 1 0/0 0 0
1 0/0 0/0 1 0/0 0 1
1 0/0 0/0 1 0/0 1 0
1 0/0 0/0 1 0/0 1 1

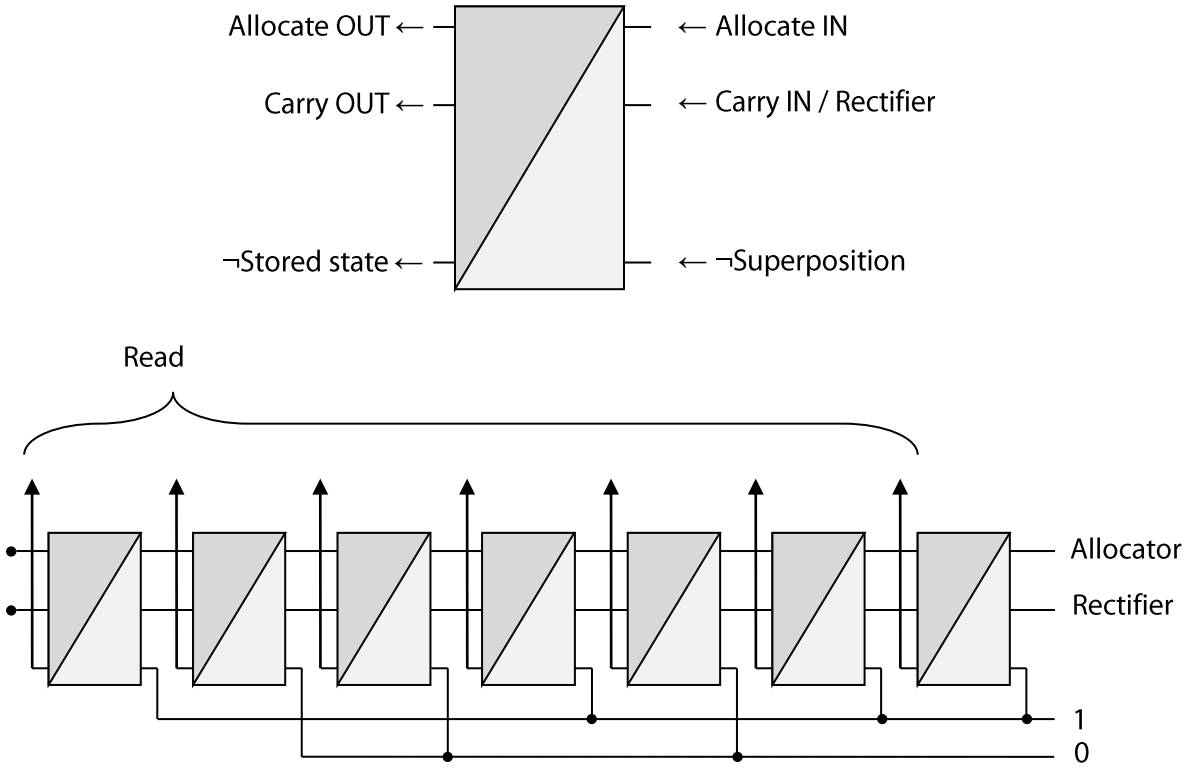
In the diagram prior, it becomes firstly apparent that the degree of symmetry is quite comically high. In this way, the diagram is a microcosm of many components of the system and the system itself, where really, symmetrical patterns are the premise of much of the design, optimisation, and analysis.

The same system can also be represented in circuitry assembly form. This truth is not of great significance to the understanding of this exact module, but rather, should be recognised in parity with the expressibility of the system. Many components of the system can be represented in ways similar to below:

(A few timing events are assumed to be correct, and for simplicity, let it be said that Allocate IN does not trigger Carry OUT, and cannot receive current other than from Allocate IN, and similarly, the superposition signal cannot feed into the flip-flop latch. The circuit does not have as its goal actual functionality, but rather, representation of functionality):



The example circuit can therefore be represented by the simple assembly:



4.13. IO_STREAM::IO_STREAM(VOID)

```
BEGIN IOStream.IOStream
    // do nothing
END IOStream
```

The IOStream constructor doesn't need to do anything, since IOStream is really just a container of interfacing methods. Different implementations of IOStream will do varying degrees of work, but most simply, the IOStream class needn't explicitly store anything.

4.14. RELATION MAIN::ADD_FACT(FACT_DATABASE DATABASE, RELATION FACT)

```
BEGIN main.addFact(factDatabase, fact)
    factDatabase.appendFact(fact)
END addFact
```

As noted in `III. 3.4., this module is probably redundant. It was envisaged in `III. 3.3. to be implemented as follows, but the design holistically appears no longer to warrant such a function --:

```
BEGIN main.addFact(factDatabase)
    fact = IOStream.getFact()
    factDatabase.appendFact(fact)
END addFact
```

4.15. RELATION MAIN::APPLY_FILTER(RELATION FACT_AGGREGATION, INT[] SYMBOLS_TO_ELIMINATE)

```
BEGIN main.applyFilter(factAggregation, symbolsToEliminate)
    Copy factAggregation into newFactAggregation
    Set numSymbolsToEliminate to number of elements in symbolsToEliminate
    FOR symbolHash = 0 TO numSymbolsToEliminate
        symbolID = symbolsToEliminate[symbolHash]
        newFactAggregation.eliminateSymbol(symbolID)
    NEXT symbolHash
    return newFactAggregation
END applyFilter
```

A wild lack optimisation can be seen; it can be speculated that factAggregation grows exponentially in proportion to the number of symbols that are to be eliminated, in parity with the implementation of the eliminateSymbol method (`4.18.). The effectiveness of optimisation in terms of the quality criteria will be explored in `III. 5..

4.16. MAIN(VOID)

```
BEGIN main.main
    // do something
END main
```

The main method does indeed have to do something. In fact, it must do something quite important, yet, it is terribly difficult to implement at this stage, and there is no benefit in attempting to do so. The other methods of the Main class are sufficient to test the system.

4.17. FRACTION MAIN::REQUEST_INFORMATION(RELATION SUBJECT, FACT_DATABASE DATABASE)

```
BEGIN main.requestInformation(subject, database)
    factAggregation = database.getFactAggregation()
    filter = IOStream.getFilter()
    factAggregation = applyFilter(factAggregation, filter)

    symbolList = factAggregation.getInvolvedSymbolList()
    Set numSymbols to number of elements in symbolList
    numConstituents = 2 ^ numSymbols

    remainingCoefficients = new Relation("1")
    vanishingCoefficients = new Relation("1")

    coefficientMap = new CoefficientMap(symbolList)
    FOR constituent = 0 TO numConstituents
        Copy factAggregation into factAggregationCopy
        factAggregationCopy.insertArguments(coefficientMap.read())

        constituentValue = ""
        FOR symbol = 0 TO numSymbols
            IF constituentMap[symbol] == 0 THEN
                constituentValue += "[", symbol, "]"
            ELSEIF constituentMap[symbol] == 1 THEN
                constituentValue += "(1 - [", symbol, "])"
            ENDIF
        NEXT symbol

        coefficient = factAggregationCopy.evaluate()
        IF coefficient == 1 THEN
            remainingCoefficients *= Relation(constituentValue)
        ELSE
            vanishingCoefficients *= Relation(constituentValue)
        ENDIF
        coefficientMap.increment()
    NEXT constituent
    E = remainingCoefficients
    EDash = vanishingCoefficients
    quatesium = new Fraction(EDash, EDash - E)
    return quatesium
END requestInformation
```

There can be observed code reuse in this module and in `4.12., so there is a possibility that these two modules can be merged -- but again, this can be considered later in the development process.

4.18. VOID RELATION::ELIMINATE_SYMBOL(INT SYMBOL_HASH)

```
BEGIN Relation.eliminateSymbol(sybolHash)
    Copy this into relationWith0
    Copy this into relationWith1
    involvedSymbolList = this.getInvolvedSymbolList()
    Set largestSymbol to the greatest value in involvedSymbolList

    Initialise argumentMap to an array of Traids, size largestSymbol + 1
    // set all symbols that aren't to be eliminated to be in the quantum
    // superposition state
    FOR argument = 0 TO largestSymbol
        argumentMap[argument] = 0/0
    NEXT argument
    argumentMap[symbolHash] = 0
    relationWith0.insertArguments(argumentMap)
    argumentMap[symbolHash] = 1
    relationWith1.insertArguments(argumentMap)
    productOfElimination = relationWith0 * relationWith1
    Copy productOfElimination into this
END eliminateSymbol
```

The allocation of largestSymbol to the greatest value of involvedSymbolList can be accomplished by implementing a simple linear search, but, many higher level languages have max(array) functions which accomplish the same goal -- there is no point detailing the logic in something that is likely to already be implemented in the source language.

4.19. BOOL RELATION::EVALUATE(VOID)

```
BEGIN Relation.evaluate
    IF this.data contains the term '[' THEN
        Display "Error: not all symbols eliminated"
    ELSE
       // evaluate this.data, and return the result ...
    ENDIF
END evaluate
```

Many languages have an inbuilt evaluate function, and the logic behind evaluation is too complex to be succinctly represented in pseudocode. A C++ implementation is attached in the appendix (`VII.) .

4.20. INT[] RELATION::GET_INVOLVED_SYMBOL_LIST(VOID)

```
BEGIN Relation.getInvolvedSymbolList
    Initialise symbolList to an empty array
    data = this.data
    Initialise symbolOccurrences to the result of splitting the string
    "data" on occurrences of '['
    // split is a standard string manipulation function, present or easily
    // implementable in most languages
    Set numOccurrences to the number of elements in symbolOccurences
    FOR occurrence = 0 TO numOccurrences
        currentSymbolString = symbolOccurences[occurrence]
        Truncate currentSymbolString at ']'
        // str.truncate(a) can be implemented as str.split(a)[0]
        Convert currentSymbolString into an integer, currentSymbol
        IF currentSymbol is not present in symbolList THEN
            Append currentSymbol to symbolList
        ENDIF
    NEXT occurrence
    RETURN symbolList
END getInvolvedSymbolList
```

The conversion of currentSymbolString to an integer can be achieved most easily by calling an evaluate function, but there are other implementations which involve the scanning of the string from right to left and summing the digits to exponents of base ten. Individual digits can be converted easily to integers by utilising ASCII values or a case statement with ten conditionals.

4.21. VOID RELATION::INSERT_ARGUMENTS(TRIAD[] SYMBOL_VALUES)

```
BEGIN Relation.insertArguments(symbolValues)
    Set numSymbols to the number of elements in symbolValues
    FOR symbol = 0 TO numSymbols
        currentSymbolValue = symbolValues[symbol]
        IF currentSymbolValue is not in the quantum state 0/0 THEN
            searchTerm = '[', symbol, ']'
            replacementTerm = '(', currentSymbolValue, ')'
            Replace occurrences of searchTerm in this.data with
                replacementTerm
        ENDIF
    NEXT symbol
END insertArguments
```

4.22. RELATION::RELATION(STRING CONSTRUCTOR)

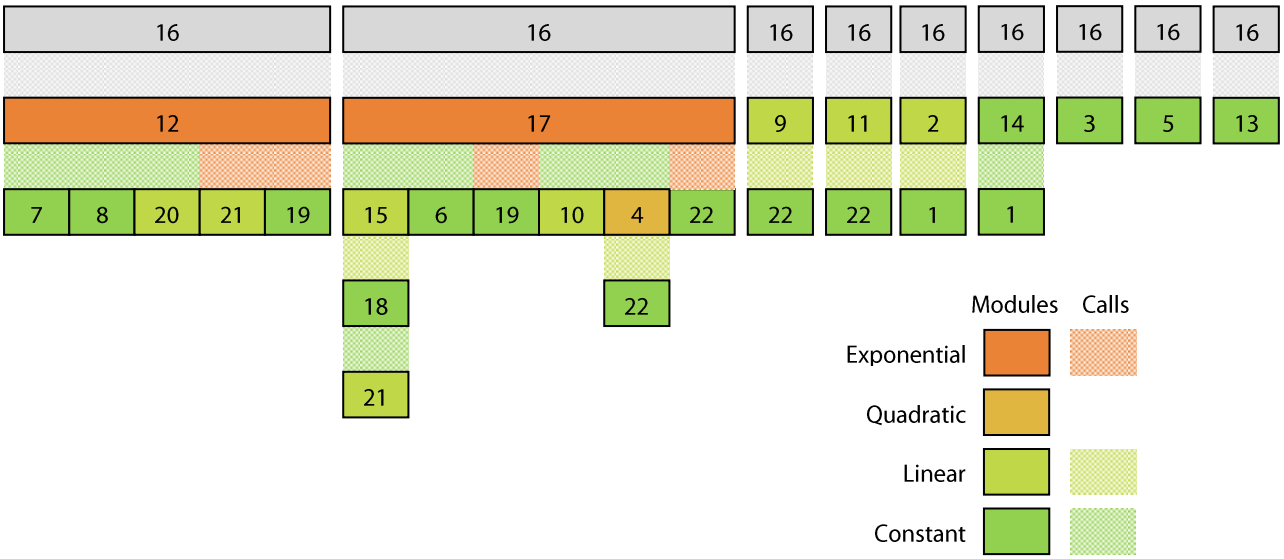
```
BEGIN Relation.Relation(constructor)
    this.data = constructor
END Relation
```

5. PRELIMINARY ANALYSIS

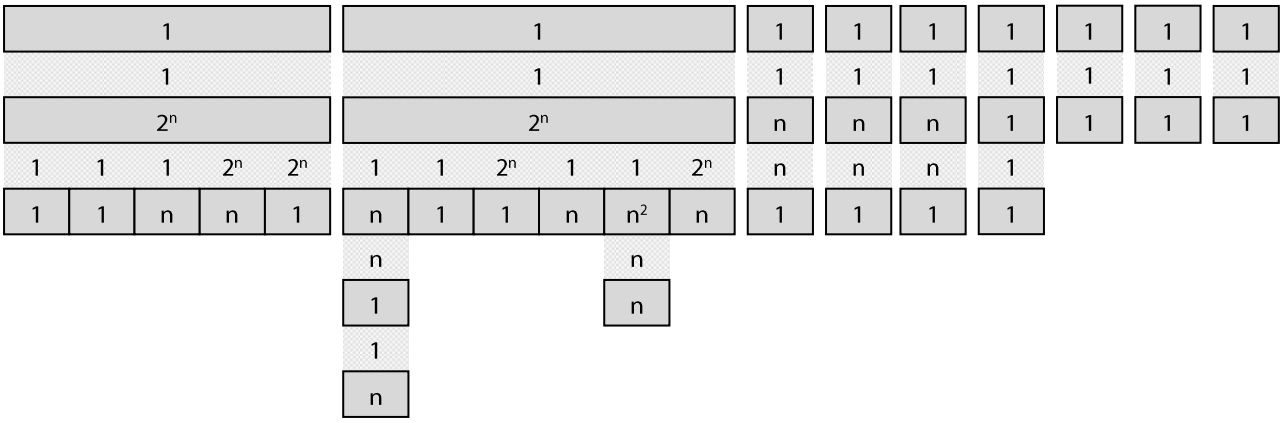
5.1. ANALYSIS & OPTIMISATION

At least exponential problem size is to be expected from this particular problem, simply because, worst case, there must be exactly 2^n coefficients and constituents printed to the screen. A time in which a solution is reached, but not necessarily displayed, is one which the developer understands to be exponential without the use of dynamic programming.

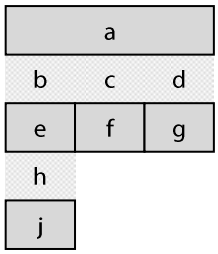
By consequence of the interdependence of modules, the efficiency of the most derived children modules must be considered firstly in order to calculate the complexity of higher level modules. The results are arranged in a call chart, which is simply a flattened structure chart (which makes many lenient assumptions about the efficiency of arrays and strings):



The numbers can be mapped to a more easily readable numerical form (especially in greyscale).



The complexity of the lowest modules are given by their contents / colour, then, the complexity of each individual of the next lowest modules equals the contents of that box plus the multiple of the lowest module and the shaded area between the modules. An example follows:



The complexity of each module in the example can be derived as next observable:

i. i

g. g

f. f

e. e + h(i)

a. a + b(e + h(i)) + c(f) + d(g)

Citing the actual graph, the complexity of modules can be calculated and arranged as follows:

1. O(1)	
2. O(n)	
3. O(1)	
4. $O(n^2)$	
5. O(1)	
6. O(1)	
7. O(1)	
8. O(1)	
9. O(n)	
10. O(n)	
11. O(n)	
12. $O(n \cdot 2^n)$	high complexity: critical path of 21, which has no dependencies
13. O(1)	
14. O(1)	
15. $O(n^2)$	
16. --	
17. $O(2^n)$	high complexity: critical paths of 19, 22, neither of which have dependencies
18. O(n)	
19. O(1)	
20. O(n)	
21. O(n)	
22. O(1)	

It is manifest that the critical paths are never anything less than exponential, so the time complexity is independent of anything that is $O(n^2)$ or less. Attention to the individual cases 12 and 17, which

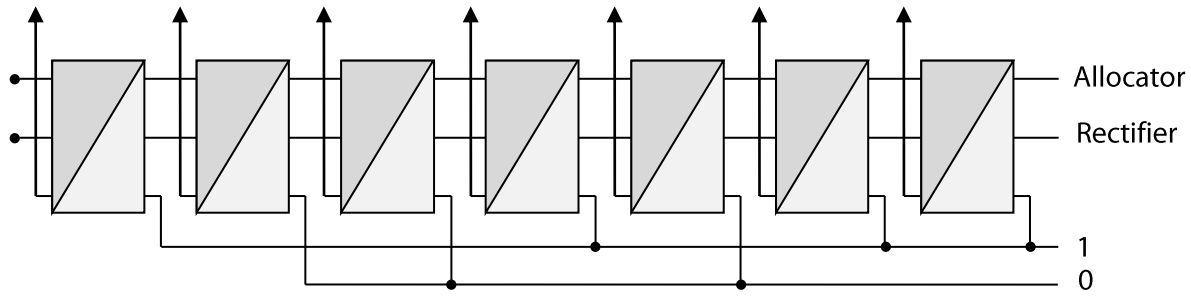
deserve their respective names giveSoughtInformation and requestInformation, reveals that they are of the same nature, namely, they are both called when a user requests information.

It has already been acknowledged that both of these functions do very much the same type of thing, which is furnished by observing their commonality in high complexity children. Module 19 can quickly be picked out as a high complexity subroutine common to both of the giveSoughtInformation and requestInformation functions. Giving this a name, it is the evaluate function. It is thus imperative that, in the chosen language, the provided or created evaluate method is made as efficient as possible.

Were the technique of dynamic programming to be employed, the coloured table quickly reveals where applications would be most auspicious, and that is, in calls from 12 to 21, 17 to 4, and 17 to 15 -- but, the developer's limited experience renders this optimisation as being detrimental to the project management wholly.

The critical path of module 17 (requestInformation) that remains to be considered (the other being evaluate) is 22, which appears at first to be extraordinarily important to optimise due to the facts that, firstly, it is called from multiple levels of function 17, with both calls being of high complexity, and secondly, it is called linearly by two separate modules. Breaching the abstraction of the number 22 reveals "relation" -- the relation object constructor. This may prematurely be disregarded as a flawed property of the call chart, since the relation constructor appears to be unoptimisable, but, more developed thought reveals that the information gained is still useful. The conclusion that can be drawn is that nothing should be added to the relation constructor unless necessary, that is to say, methods such as input validation or compression should be done externally.

What should be considered as being the final contributor to the two critical paths is module 21, insertArguments, belonging to the Relation class. This module is called repetitively on the same Relation class from module 12 (giveSoughtInformation) with common arguments, particularly, the arguments only differ by the incrementer input, recalling the following example of the argument arrangement:



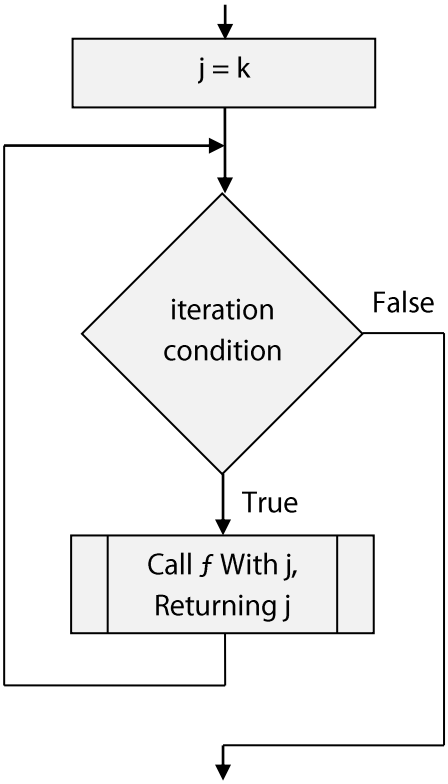
It would also be beneficial, since this method is being invoked on Relations which are simply throwaway copies of one base class, to do one single priming read to gather character position information to be used in all 2^n repeats of this routine, which would reduce the complexity of the insertArguments module to be linear to the number of arguments, as opposed to the length of the string that is being parsed. The implementation costs of this may however be excessive.

Conclusively, the main factor in terms of operational efficiency is the choice of the correct algorithm wholly. Secondary to this comes the selection of appropriate algorithms for low level management, such as the multiplication of Relations, and low-priority optimisation can also be found in a few key areas. In its unoptimised state, the algorithm runs wholly at $O(n \cdot 2^n)$, but with some consideration, this can easily be reduced to $O(2^n)$, but perhaps no further without specialised dynamic programming techniques or the divinity of a better algorithm. The speed of the program is majorly proportional to the number of symbols involved in relationships between facts, but its speed still has significant grounding in the number of facts and their complexity. Despite, however, the obvious advantages of optimisation, the implementation should not be founded in their exact consideration.

5.2. PARALLELISATION

It is plain to see that the program repeats a series of tasks over and over again an exponential amount of times. There are two general scenarios where this repeated processing can be seen:

(i) A process is called recursively; each iteration of the loop takes results dependant on the previous iteration, and that data becomes the argument of a complex process to find the result of the iteration, and then, the process is repeated:



After one iteration, it should be noticed that j equals $f(k)$, two iterations implies $j = f(f(k))$, three iterations $f(f(f(k)))$, and that this result is can be extrapolated. Assuming f is a complex function, parallelisation in this scenario is quite difficult.

(ii) A process is called linearly, and the result does not in any way affect the next computation. A sequence of results are created, and they are only merged after the loop has been executed. The result of each iteration is entirely independent of previous results, and the program can be parallelised.

It is the second scenario that is obviously present in the problem of a P.L.E: the exponential routines call either independent output functions, or construct multiples of a series of generated coefficients. The efficiency of parallelisation can now be analysed to provide connexion between the parallelisable portions of the program, the number of processors available for utilisation, the number of symbols, and a speedup.

Let it be considered a time in which the main modules (where most of the complexity is founded), 12 and 17, are executed, respectively, giveSoughtInformation and requestInformation. Their respective complexities can be calculated from the call chart:

$\Theta_{12} = (n + 1)2^n + n (+ C);$
$\Theta_{17} = 2 \cdot 2^n + 2n^2 + 2n (+ C);$

Quite clearly, if the exponentiality of both routines are not distributed, the runtime of the algorithm will remain exponential, achieving, at best, a 2 times speedup, even with an infinite amount of processors (by Amdahl's Law). Whence, it can be deduced that the only scenario where optimisation is worth anything is when both sets of 2^n can be reduced. With an infinite number of cores to distribute the workload across, the actual running time in this optimal scenario becomes quite small:

$\Theta = 2n^2 + 4n (+ C)$

Analysis also reveals that the ratio between cores and runtime in the situation of full parallelisation is approximately linear, as summarised below:

Cores	~Processing Time (Static n)	Efficiency Gain (Repeated Differences)
0	∞	--
1	100%	∞
2	50%	50%
3	33.3%	16.7%
4	25%	8.33%
5	20%	5%
6	16.7%	3.33%
7	14.3%	2.38%
8	12.5%	1.79%
9	11.1%	1.39%
10	10%	1.11%

One additional result must importantly be drawn: to keep runtime constant, for every new symbol, the number of cores must be doubled. This is the primary bounding factor for parallelisation.

5.3. THE INTERCHANGABILITY OF MEMORY AND STORAGE

The time complexity of the program isn't the only thing that is exponential, so too is memory usage. From the client's machine specification, there is only 5.65 GB of available RAM. This, certainly, is practical for most uses of the program, but the question becomes whether the operability of the program is bounded firstly by available memory, or whether it is bounded by time.

It is important to recognise that the data doesn't have to be loaded into RAM, since at any given time, only a very small portion of it is being operated upon (in most circumstances). Thus, if the performance of the program is bounded by memory and has time to spare, it can offload the data in RAM to virtual memory, or explicitly into secondary storage.

IMPLEMENTATION

PART IV.

1. SOLUTION STRUCTURE

1.1. LANGUAGE & DESIGN APPROACH

There exist many different types of languages and various approaches in actual application development, but mainly, both have as primary influences the problem's nature. Python 3 will be the language of implementation by the allowance of three factors: the developer's experience in it, the broadly defined running environment, and, indeed, the nature of the problem. In terms of nature, the problem is well suited to Python firstly because Python offers the necessary functionality, and secondly because of the simplicity and level of abstraction of the solution: there needn't be much machine-level interaction, such as explicit memory management, which escapes the common functionality of Python. The GUI component will run in a JavaScript / HTML / CSS emulator within Python.

The design approach, broadly, will programming occurring from the ground up, namely, a bottom-up approach. This is complementary to the top-down approach that was used in defining the problem: now that it has become known what exactly needs to be done, the individual constituents of the problem can be handled in their simplest components. The reason for this necessity lies in the form of what will be developed, indeed, the first round of implementation will see a program generated without an interface to the outside world. All the program will do is print either "all tests passed", or "tests failed", which is in no way shippable. At the point in which unit tests have been produced, and the output is "all tests passed", then all of the smaller components can be linked together with a GUI or CLI. It is certainly not a good idea to design a GUI first, which would be a trait of a top-down approach, since it would have to be iteratively modified when features are inevitably forgotten, or newly added.

1.2. DIRECTORY STRUCTURE FOR DEVELOPMENT

In correlation with the designed algorithm (plus the additional quantum counter theorised in `III. 4.12.), the directory structure for initial development follows (and is generalisable for any additions):

```
PLE
│   testPLE.py
│
├───FactDatabase
│       FactDatabase.py
│       testFactDatabase.py
├───Fraction
│       Fraction.py
│       testFraction.py
├───QuantumCounter
│       QuantumCounter.py
│       testQuantumCounter.py
└───Relation
        Relation.py
        testRelation.py
```

The lack of a main module is due to this being the development directory structure, not the final one. Actual user interaction is not an early consideration, and will not be present until the late stages of development, where a main module can be included as part of new IOStream packages. As is, despite there being a testPLE.py module, there is no PLE class.

1.3. CASE TOOLS

Visual Studio 2015 (Community), by the benevolence of Microsoft, brings project file management from near impossible to quite easily maintainable. Visual Studio also provides a smorgasbord of useful debugging and analysis tools, and thus has a great advantage over Python's IDLE, which doesn't even support the concurrent displaying of multiple source files. CASE tools are necessary in the development of a solution package, and several specifically are of great use.

IntelliSense, which is a form of code completion and interface / type linking across multiple files, is one of the first notable benefits of Visual Studio. IntelliSense offers concurrent suggestions while typing, and provides information about particular modules or variables on mouse hover events, and additionally, the IDE provides refactoring tools and some automatic code generation to further aid in the development of source code.

Visual Studio offers a comprehensive debugging environment, where breakpoints can be set conditionally or unconditionally, stepped over, have an execution path be entirely changed, and, variables can be examined and modified during debugging runtime. There is also an ineffable amount of tools available for performing diagnostics and code analysis.

The actual Visual Studio environment is also quite impressive, whereby the GUI is highly configurable and provides helpful information through things such as the solution explorer, which is an internal means through which to manage project files. Other significant GUI tabs include an interactive Python shell, class view, performance explorer, and many others.

2. MODELING OVERVIEW

2.1. RELATION STORAGE

Relations, in the form that follows, can be stored in two primary ways, both of which carry simultaneous advantages and disadvantages in implementation:

$\delta(\theta_0, \theta_1, \ldots \theta_n) = 0;$

This information can either be stored through the connexion of symbols, or as a series of binary flags to represent the coefficient of each constituent. The two alternatives of a coefficient map structure or simplified expression storage follow, where [0] and [1] respectively mean θ_0 and θ_1, in the development $\delta(\theta_0, \theta_1) = 0;$

Possible State	Coefficient Map				Simplified Expression
	[0][1]	[0](1-1])	(1-[0])[1]	(1-[0])(1-[1])	
1	0	0	0	0	0
2	0	0	0	1	(1 - [0])(1 - [1])
3	0	0	1	0	(1 - [0])[1]
4	0	0	1	1	(1 - [0])
5	0	1	0	0	[0](1 - [1])
6	0	1	0	1	(1 - [1])
7	0	1	1	0	[0](1 - [1]) + (1 - [0])[1]
8	0	1	1	1	[0](1 - [1]) + (1 - [0])
9	1	0	0	0	[0][1]
10	1	0	0	1	[0][1] + (1 - [0])(1 - [1])
11	1	0	1	0	[1]
12	1	0	1	1	[1] + (1 - [0])(1 - [1])
13	1	1	0	0	[0]
14	1	1	0	1	[0] + (1 - [0])(1 - [1])
15	1	1	1	0	[0] + (1 - [0])[1]
16	1	1	1	1	1

It can be seen that both the simplified expression, and the string of four Boolean digits in the coefficient map encapsulate the same information, but both do it very differently. Relations can evidently be stored either mathematically or symbolically, and it is left that both ways are discussed.

If a Relation is stored as a coefficient map, for example, 0110, much of the information is migrated (hence the necessity of a map). This means that there is a large overhead when it comes to performing operations on individual Relations, but, there are many shortcuts available when performing similar operations on many Relations. Storage as such also greatly reduces memory usage wholly.

Storing a Relation as a simplified expression, despite being certainly non-optimal in terms of speed and memory usage, is much easier to implement and get correct, since physical memory interactions don't need to occur. The current program state will have this implementation, since

testability and ease of implementation take precedence over efficiency. With the help of abstract interfaces, Relation can be re-implemented in the future.

2.2. QUANTUM_COUNTER MODEL

Recalling `III. 4.12., the QuantumCounter class requires a list of superposition states to be constructed. This can be done several ways:

(i) Providing the number of bits and an array of qubit positions (or an array of non-qubit positions).

```
new QuantumCounter(7, [1, 2, 4])
```

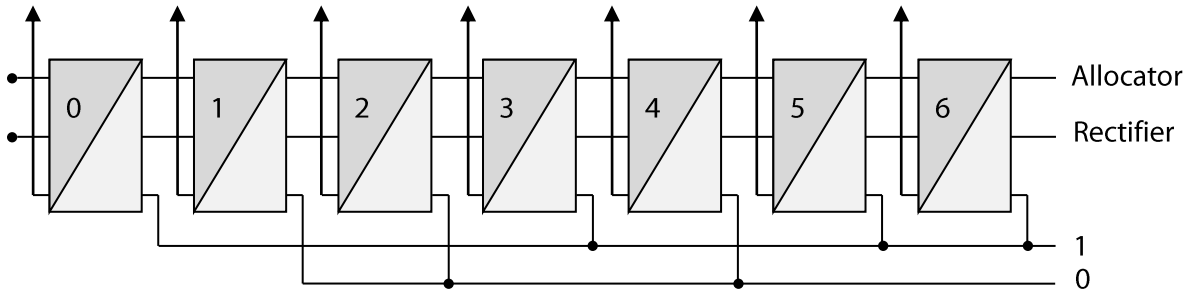
(ii) Providing an array of both qubit and bit positions.

```
new QuantumCounter([1, 2, 4], [0, 3, 5, 6])
```

(iii) Providing an array representing the bit-qubit state of each respective bit with Boolean flags.

```
new QuantumCounter([0, 1, 1, 0, 1, 0, 0])
```

Method (iii) is more in line with the model from `III. 4.12., where each element of the array has a value according to the two instantiation wires, labelled 0 and 1 but notice the inversion, and the bits are indexed from left-to-right (as in the source).



This model of construction is understandably easier, since it allows both the constructor to internally generate a correct object with more ease, by virtue of the arguments being more representative of physical phenomena, and also allows the caller to the constructor to provide the required arguments, since an array of Boolean values is quite easy to create. It is also extensible so as to allow a QuantumCounter to be instantiated with non-zero initial values, as can be achieved by representing 0 states as 0, 1 states as 1, and superpositions as any other value. It is thus that the signature of the function accepts an array of 1's and 0's.

3. CORE VARIABLE DOCUMENTATION

3.0. NOTES

It must be noted that much of the documentation exists inherently in the manual (external to this document), as done in attempt to prevent repetition. Until the manual is read (and in saying this, the necessity of quality is overlayed on the manual), it may be felt that the solution is not sufficiently documented, and especially not in a way that is understandable to the end-user.

This section (and its adjacency) also doubles as an incomplete technical manual. Were the need for a technical manual apparent -- which it so manifestly isn't, since it would mostly be a summary of this report whole (although pithy), and much documentation exists internally in the file (with the aids of comments). The necessity of a technical manual would only for maintenance purposes, but, internally documented source code and this report itself serve to the same end, and thus, a technical manual will not be included within this report package.

3.1. DATA DICTIONARY

For brevity, descriptions are as pithy as possible (since the data dictionary is mostly superfluous to the end of maintenance). Several columns that may be standardised can be seen as missing, as justified:

- "Format" rows are missing, since most data items formats are either intuitive, type-dependant, or are unbounded. Where necessary however, format has been merged into the description.

- "Number of bytes" remains unlisted because Python is not a strictly typed language, so variables physical sizes can grow and shrink in accordance with their contents. This heading wouldn't provide much useful information regardless.

- "Example", despite being irrefutably beneficial in some circumstances, is omitted for brevity, as it would rarely be filled with expressive information. In particular circumstances, if an example is necessary, it finds its place in the description column.

Items are listed alphabetically in piles based on source files (only the core source files), and capital letters take precedence over lowercase. Naming collisions are resolved by giving function context.

FactDatabase.py		
Data Item	Type	Description
UNITY	Relation (Record)	Boolean representation of the universe. Used to evaluate the complement of other *Relations* -- for example, "not [0]" can be calculated as "*UNITY* - [0]".
aggregate	Relation (Record)	The aggregate that is returned by *getFactAggregation* -- the combined information of all facts within a *FactDatabase*.
currentFact	Relation (Record)	Fact with an index correlating to *factNumber*.

…

Data Item	Type	Description
fact (appendFact)	Relation (Record)	Argument provided by caller. The fact that is to be appended.
fact (getFactAggregation)	Relation (Record)	Range iterator -- stores the next fact to be optimised in the list of facts held in *FactDatabase*.
factNumber (deleteFact)	Integer	Parameter to specify the fact which the caller wishes to delete.
factNumber (getFactAggregation)	Integer	Iterator. Holds the index of the fact that is next to be processed in order to generate *aggregate*. Bounded by the length of *optimisedFactList*.
numFacts	Integer	The length of *optimisedFactList*. Boundary for the incrementation of *factNumber*.
optimisedFactList	Array of Facts (Records)	An optimised copy of the data within a *FactDatabase*, so as to allow the generation of a representation of all facts (*aggregate*) in a way that saves memory.
priorFactNumber	Integer	Iterator -- bounded by 0 and *numFacts*. Holds the index of facts that are to have their complements taken, then be multiplied into *productOfPriorFacts*.
productOfPriorFacts	Relation (Record)	Product of complements of facts that have already been iterated over, namely, facts with an index less than *factNumber*. Multiplied with *currentFact* and added and stored in *aggregate* at the end of each iteration of *factNumber*, thus generating *aggregate* wholly.
successState (appendFact)	Boolean Value	Stores whether or not a fact append operation was successful. Contains False if the fact is malformed, and could therefore not be appended. Returned to the caller as a flag.
successState (deleteFact)	Boolean Value	Stores whether or not a fact that was targeted for deletion is in range, and is therefore deletable (and then deleted). Returned to the caller as an indicator for success.

Fraction.py		
Data Item	Type	Description
numerator	Relation (Record)	Argument to construct a Fraction: the numerator of the tuple.
denominator	Relation (Record)	The second argument needed to construct a Fraction, being the denominator.

Interface.py		
Data Item	Type	Description
aggregate	Relation (Record)	Representation of all factual data within the system.
coefficientMap	QuantumCounter (Record)	A map of parameters / arguments with which a *Processor* is constructed, and thus, with which the Processor iterates. Set to the first coefficient arrangement in *request (generateProcessor)*.

…

Data Item	Type	Description
denominator	Relation (Record)	The denominator with which to construct a *Fraction* to be returned by *_aggregateResponse*, and then, returned by *retrieveResponse*. According to Boole, t = E' / (E' - E) (as furnished in `III. 1.2. of this report), whence, E' - E is the denominator (*productWith0* - productWith1).
errorOutputMethod	Function	Method provided as an argument to the Interface constructor to specify how various errors are to be handled. If no argument is provided, this item defaults to containing a function which suppresses all error messages.
fact	Relation (Record)	Parameter in *addFact*, specifying the *Relation* that is to be appended to an internal *FactDatabase*. If the data is malformed, an error is raised through *errorOutputMethod*.
factNumber	Integer	Index corresponding to a fact that the caller wishes to have deleted.
handler	Function	Method to cause a *Processor* to run. When evaluated, allows an *Interface* to be queried for a response through the *retrieveResponse* interface function. Returned by *generateProcessor*.
isValid	Boolean Value	Stores whether or not the *request (setRequest)* which the caller is trying to set is malformed. Raises an error if the flag is set to False.
numerator	Relation (Record)	The denominator with which to construct a *Fraction* returned eventually by *retrieveResponse*, after first being bubbled through the *_aggregateResponse* helper function. According to Boole, t = E' / (E' - E) (see `III. 1.2. of this report for a proof), therefore the numerator is E', which is symbolised by *productWith0*.
productWith0	Relation (Record)	Categorically half of the information that a *Processor* generates. The product of coefficients in the full development whose respective coefficients in the symbol's development are in it for zero. Termed by Boole as E' (see *numerator* or *denominator*).
productWith1	Relation (Record)	One of the two results of reading the output from a finished *Processor*. The product of coefficients in the full development whose correlating coefficients in the symbol's development are equal to unity. Termed by Boole as E (see *numerator* or *denominator*).
repository	Processor (Record)	The completed processor whose results are to be read in order to construct a *Fraction* encapsulating the response of the PLE system to a query. Must have finished processing, as validated by the *retrieveResponse* interface function.
request (generateProcessor)	Relation (Record)	The return value of the *getRequest* member function. The request which the user has specified, which will be used to construct a *Processor*.
request (setRequest)	Relation (Record)	Specifies the request which a *Processor* will be developed according to in *retrieveResponse*.

…

Data Item	Type	Description
request (getRequest)	Relation (Record)	The request that corresponds to *request (setRequest)*. Returned to the caller to be observed. If the request is undefined, this variable contains None.
requestSymbolList	Array of Integers	A list of symbol keys that are present in *request (generateProcessor)*. Argument to the *QuantumCounter* method *createCoefficientMap*, to construct a suitable QuantumCounter for incrementation (*coefficientMap*).
returnValue	Fraction (Record)	*Fraction* to be returned by *retrieveResponses*. Contains None if either no processor has been generated, or if the *Processor* has not completed its processing. (By *Processor* is specifically meant the return value of *generateProcessor*)
successState (addFact)	Boolean Value	Flag to indicate whether or not the *fact* parameter to *addFact* was successfully appended. If set to False, an error will be thrown through *errorOutputMethod*.
successState (deleteFact)	Boolean Value	Bit to store the success of fact deletion in *deleteFact*. If this bit is set to False, an error will be raised by *errorOutputMethod*.
totalWork	Integer	The number of iterations to *coefficientMap* that must be made before the *QuantumCounter* overflows. Needed to construct a *Processor*.

Processor.py		
Data Item	Type	Description
aggregate	Relation (Record)	The combined knowledge of all information within the PLE, with which to construct the *Processor*.
coefficient	Relation (Record)	The coefficient that specifies whether to multiply *fullDevelopmentCcopy* into *productWith0* or *productWith1*. The result of evaluating *soughtDevelopmentCopy* with the *QuantumCounter* iterator beginning at *quantumCounter*.
finishedProcessing	Boolean Value	Flag to indicate whether or not the *Processor* has a ready result -- more specifically, set to true if and only if the *go* method has returned.
fullDevelopmentCopy	Relation (Record)	A copy of the locally stored *aggregate*, which will be fed a unique incremented value of *quantumCounter*. Once the arguments in *quantumCounter* have been inserted, this data items value is assigned to *coefficient*.
numIterations	Integer	The number of times *quantumCounter* must be incremented before it overflows, that is, the number of constituents in *soughtDevelopment* (which is 2 raised to the power of however many symbols there are in *soughtDevelopment*).
quantumCounter	QuantumCounter (Record)	The first coefficient map with which to evaluate copies of *aggregate* and *soughtDevelopment*. A copy of this argument is iterated *numIterations* times within the *go* method.
soughtDevelopment	Relation (Record)	It becomes the goal of the processor to solve for this constructor variable, using the information of *aggregate*.
soughtDevelopmentCopy	Relation (Record)	Stores a copy of *soughtDevelopment* which is to be manipulated such that it becomes evaluated with *quantumCounter* to generate *coefficient*.

QuantumCounter.py		
Data Item	Type	Description
bitNumber (__init__)	Integer	Iterator to keep track of which bit in *superPositionStates* is being read (corresponding to a write in the member array *bitStates*). Set initially to zero, and bounded by the length of *superPositionStates* (as stored in *numBits*).
bitNumber (increment)	Integer	Iterator (that gets decremented). A pointer to the rightmost unprocessed bit in the *bitStates* member variable.
bitNumber (toString)	Integer	Iterator -- for each item in the member variable *bitStates*. Allows the traversal of the array from left to right, appending some information to *string* for each item. Bounded by *numBits (toString)*.
carry	Boolean Value / Integer	An integer that takes the value of either 0 or 1 (in the implementation, represented as an Integer, but a Boolean data type accomplishes the same result). Stores whether or not to carry in (equally, what value to carry in) when performing an addition operation. Set to 1 initially since the purpose of the routine is to increment the *QuantumCounter* state. Contains 1 while no 0s have been encountered in the member variable *bitStates* (as stored in *currentState*).
coefficientArray	Array of non-typed Objects	Used to construct a *QuantumCounter*, namely, the argument that maps to the parameter *superPositionStates* in the *__init__* method. Is set to hold values such that there is a quantum state for every element not present in *involvedSymbolList*.
coefficientMap	Record (QuantumCounter)	A local *QuantumCounter* constructed with *coefficientArray*. A hacky variable to prevent the need for manual construction of a *QuantumCounter*; allows the already written *__init__* method to be used.
currentBit	non-typed Object	The element under observation in the member variable *bitStates* -- the *bitNumber*-nth *(toString)* element of the array. Used to conditionally append information to *string*.
currentState	non-typed Object	The *bitNumber*-nth *(increment)* element of the member variable *bitStates*.
involvedSymbolList	Array of Integers	A list of symbols (which are involved in some abstract *Relation*), with which, it is purposed to initiate a *QuantumCounter*, which, for each of these symbols, has an indexed value of 0, and otherwise, is in a quantum superposition.
iteration	Integer	Iterator to enact the process "simulate *numIteration* iterations". Set to 0 initially, and incremented in correspondence with the *QuantumCounter* object, until bounded by the argument *numIterations*.
largestSymbol	Integer	The largest symbol in *involvedSymbolList*. This is needed to derive the necessary length of *coefficientArray*.
numBits (__init__)	Integer	Stores the number of bits in *superPositionStates*, and thus, the desired length of the member *bitStates*.
numBits (toString)	Integer	Holds the length of the member *bitStates*.
numIteration	Integer	Parameter to confer the amount of times a caller wishes to simulate incrementation of the *QuantumCounter* instance.

…

Data Item	Type	Description
source	Record (QuantumCounter)	The source *QuantumCounter* in a copy operation, namely, the *QuantumCounter* that is to be copied into the object whose method is being called.
string	String	Return value of *toString*: a representation of a *QuantumCounter* instance in string form. Set initially to an empty string, but information can be concatenated in one of the two forms "(1 - [n])" or "[n]", where n is an integer, depending on the value of *currentBit*.
superPositionStates	Array of non-typed Objects	Comprised of elements categorised as either 0, 1, or a quantum state (anything else). Used to construct a *QuantumCounter*. Indexes which correspond to values of 0 will be initiated to a default value of 0, and likewise for 1, but for any other state, the local *bitStates* member's index gets set to a value indicative of a quantum state, which will be passed over during iteration. Empty by default.
symbol	Integer	Range Iterator -- an element of *involvedSymbolList*. Used to initiate the values in *coefficientArray* with indexes *symbol* to 0.

Relation.py		
Data Item	Type	Description
LIST_OF_ILLEGAL_EXPRESSIONS	Array of Strings	An array of character combinations that purposes to list every possible type of illegal marriage. If any illegal combination is found in *reducedData*, the *hasFoundIllegalCombination* flag is set to True, which bubbles up through several processes.
arg (__add__)	Record (Relation)	Together with the *Relation* instance who owns the method which is being called, comprises the two parameters needed to enact binary addition.
arg (__eq__)	Record (Relation)	The second argument in the equality comparator, the first being the *Relation* instance who owns the *__eq__* method.
arg (__mul__)	Record (Relation)	Specifies what the caller wishes to be the second argument in a multiplication operation, the first being the data stored in the instance whose method is being actuated.
arg (__ne__)	Record (Relation)	Argument which is to be compared with the scoped *Relation* instance for inequality.
arg (__sub__)	Record (Relation)	Parameter for the subtraction operator, namely, the parameter "B" in the mathematical expression "A - B".
argumentMap	Array of Integers	*quantumCounter* in array form. See *quantumCounter*.
argumentNumber	Integer	Iterator between 0 and *numArguments*. Allows each argument to be inserted into its parameter in the member string *data*.
character (_areAllCharactersLegal)	Character	Range iterator -- the characters in *dataString (_areAllCharactersLegal)*. Allows each character to be checked in the string so as to allocate an appropriate value to *hasFoundIllegalChar*.
character (_areAllNumbersLegal)	Character	Range iterator. Increments over each character that comprises the string *dataString (_areNumbersLegal)*, allowing each character to be processed if it is a digit, thus forming the *involvedNumbers* array.

…

Data Item	Type	Description
character (_isNestingLegal)	Character	Range iterator for every character in *datastring (_isNestingLegal)*. If this variable takes the form of a recognisable bracket, *listOfBrackets* has this value appended.
coefficientMap (__eq__)	Record (QuantumCounter)	Set to the first permutation of coefficients in a *Relation* consisting of the symbols in *symbolList (__eq__)*. Incremented *numCoefficients* times such that the two *Relations* that are being compared for equality can be evaluated with all possible combinations of arguments.
coefficientNumber	Integer	Iterator, bounded by *numCoefficients*. This variable serves solely to cause the comparison loop to be executed the required amount of times (such that each coefficient in the two *Relations* have been compared). Correlates to the number of times *coefficientMap (__eq__)* has been incremented.
constructor	String	The *Relation* in string from, such as "[0](1 - [1]) + [2](1 - [0])". This data must adhere to a strict set of rules to be valid, otherwise, errors will be raised elsewhere (the validity is checked by *isValid*). If left empty, it becomes the responsibility of the caller to initiate the *Relation* through the *copyFrom* method. Initiates the local member *data*. The EBNF for a valid *constructor* appears in the manual.
currentArgument	non-typed Object	The value of *argumentMap* indexed at *argumentNumber*. If this value is 0 or 1, a replacement will occur.
currentNumber	Integer	Iterator -- bounded by the number of *characters* (*_areNumbersLegal*) in *dataString (_areNumbersLegal)*. Keeps track of the index which is currently being looked at in the first loop of the routine.
currentSymbol	Integer	See below.
currentSymbolString	String	Adorned element of *symbolOccurrences* that is ready to be converted to an integer and appended to *symbolList* if it is not already found to be present in the array. Implicitly converted into an integer, taking the name *currentSymbol*.
dataString (_areAllCharactersLegal)	String	A copy of the member variable *data* for shorthand reference.
dataString (_areCombinationsLegal)	String	See above.
dataString (_areNumbersLegal)	String	A copy of the member variable *data* for shorthand reference (as restated above).
dataString (_isNestingLegal)	String	See above.
digit	Character	A digit. Range iterator. Respectively takes the stringified values 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, such that their occurrences can be replaced with the single character 'n' in the reduction *reducedData*.
evaluableString	String	A copy of the intrinsic member variable *data* in a manipulated form such that the Python function *eval* can process it.

…

Data Item	Type	Description
expression	String	Range iterator -- an indexed value in the array of illegal combinations of characters *LIST_OF_ILLEGAL_EXPRESSIONS*. Allows each individual expression to be searched for in the string *reducedData*, so that *hasFoundIllegalCombination* can be allocated correctly.
hasEncounteredDiscrepancy	Boolean Value	Boolean value to flag whether or not, in comparing each individual coefficient of a *Relation*, there has been a mismatch. If there is a mismatch, the equality operator *__eq__* returns False.
hasFoundIllegalChar	Boolean Value	Flag to indicate that a character in *dataString (_areAllCharactersLegal)* is not present in the list of legal characters "0123456789 +–()[]".
hasFoundIllegalNum	Boolean Value	The return value of the function *_areNumbersLegal* (after an inversion takes place). Set to True if a *number* has as its first indexed character a '0', and is not wholly "0" -- namely, whether it has any leading zeros.
involvedNumbers	Array of Strings	Made to contain a list of all numbers in a *Relation*. A *Relation* that was constructed with the string "00 + 01(1 - [200])" will yield the value ["00", "01", "1", "200"]. This is then searched for leading zeros, to set the *hasFoundIllegalNum* flag.
isBalanced	Boolean Value	Stores whether or not *unelminatedBrackets* has been reduced to an empty string. An empty string means that the argument *dataString (_isNestingLegal)* is balanced, and this variable takes its value accordingly.
isValid	Boolean Value	The return result of the *isValid* function. Stores true if and only if each of *_areAllCharactersLegal*, *_areCombinationsLegal*, *_isNestingLegal*, *_areNumbersLegal* return True, and if *dataString* is subjected to what Boole occasionally calls the "law of duality", n(1 - n) = 0.
listOfBrackets	String	*dataString (_isNestingLegal)* with all non-brackets removed, taking values such as "(((()(()))()()))(()()()(()((()))))(()))". This string will then be reduced via the *_recursivelyReplace* method, where legal pairs of brackets are eliminated. This allows the generation of *uneliminatedBrackets*.
numArguments	Integer	The number of arguments that must be inserted (or iterated over and not inserted). The length of *argumentMap*.
numCoefficients	Integer	Stores the number of unique coefficients there are in an abstract *Relation* with *numInvolvedSymbols* parameters. Corresponds to the amount of times *coefficientMap (__eq__)* must be incremented so that every possible permutation of the *QuantumCounter*'s bit-states is made to be exhausted.
numInvolvedSymbols	Integer	The number of unique parameters there are in the locally stored *Relation* and *arg (__eq__)*. Example: "[1](1 - [0])" and "[0](1 - [3])" have the combined symbol list of [0, 1, 3], and thus, this is the value that would be taken by this variable. Used to calculate *numCoefficients*.
numOccurrences	Integer	The length of *symbolOccurrences*, and therefore, the number of times *occurrence* must be iterated for each item to be processed.

…

Data Item	Type	Description
number	String	Range iterator over the *involvedNumbers* field. Represents each number that is to be compared against a simple conditional to raise *hasFoundIllegalNum*.
occurrence	Integer	Iterator for each respective index of *symbolOccurrences*.
quantumCounter	Record (QuantumCounter)	A map of arguments which are to be inserted into the *Relation*. Values of this array that are neither 1 nor 0 cause no change, whereas values of 1 or 0 cause all symbols with the ID *argumentNumber* to be changed accordingly.
reducedData	String	A copy of the argument *dataString (_areCombinationsLegal)* which is to be reduced by a set of rules. If this variable is successfully reduced to a string consisting only of valid expressions, namely, those which are not mentioned in *LIST_OF_ILLEGAL_EXPREPRESSIONS*, True can be returned by the function *_areCombinationsLegal*.
relationA	Record (Relation)	A temporary copy of one of the two arguments in comparison. RelationA is symmetrical with RelationB. Exists to be evaluated with the arguments in *coefficientMap*, and then compared with its namesake. Any mismatch causes the flag *hasEncounteredDiscrepancy* to trigger, thus resulting in a failed equality test.
relationB	Record (Relation)	See above.
relationWith0	Record (Relation)	The result of evaluating the *Relation* instance (which is being called) with the argument *symbolNumber* set to 0. Used to generate *resultOfElimination*.
relationWith1	Record (Relation)	The result of evaluating the *Relation* object whose *eliminateSymbol* method is being called with the argument *symbolNumber* set to 1. Used to generate *resultOfElimination*.
replacementArgument	String	The string contents that are to replace all occurrences of *symbolToReplace* in the member variable *data*. Enacting all replacements is the goal of the *insertArguments* routine.
replacementTerm	String	Parameter specifying what *searchTerm* is to be recursively replaced with in the array of characters *string*.
resultOfElimination	Record (Relation)	The result of eliminating the symbol *symbolNumber* from the invoking *Relation*. This variable is the product of *relationWith0* and *relationWith1*, symbolic of the functionality of unknowns being in quantum states, where the truth and falsehood of something is concurrently assumed. This is the final result of the routine, and will be copied into the *Relation* instance who owns the method which is being called.
searchTerm	String	The term which is to replace *replacementTerm* in *string* during a recursive replacement. VALIDATION: an error will be thrown if either the condition that *searchTerm* equals *replacementTerm* is satisfied, or the string which is to be searched is empty; provided in both cases that *replacementTerm* contains *searchTerm*. Operations with non-conformant data results in an infinite loop.

…

Data Item	Type	Description
source	Record (Relation)	Specifies the address from which a copy is to be performed. The contents of this Relation are copied into the internal *data* member variable.
string	String	The parameter in the *_recursivelyReplace* function through which the caller specifies what the data is that is to have a search and replace operation performed on it (recursively).
stringAfterChange	String	A copy of *string* (that compounds upon itself) that has undergone some number of replacements of *searchTerm* with *replacementTerm* according to the loop in *_recursivelyReplace*. This value is compared with *stringBeforeChange*, and if there is no disparity, it is manifest that the replacement process is complete, and thus, this value is returned.
stringBeforeChange	String	A copy of *stringAfterChange* that is one iteration behind. Keeping this information allows it to be checked whether or not a search and replace operation has had any effect, thus allowing the loop in *_recursivelyReplace* to terminate when the process is finished. Initially set to an empty string, since, if *stringAfterChange* is at any point empty, then indeed, the behaviour of termination is what is sought.
symbolList (__eq__)	Array of Integers	The list combined list of symbols in *arg (__eq__)* and the local member *data*. Contains no repeated items. Used to construct *coefficientMap (__eq__)*.
symbolList (getSymbolList)	Array of Integers	List of symbols involved in a *Relation* instance, which is to be returned via *getSymbolList*. Begins as an empty array. The *symbolOccurrences* array is then passed through linearly, and any encountered items not already in this list are appended, and the array is thence filled.
symbolNumber	Integer	Specified by the caller -- the symbol ID which is to be eliminated.
symbolOccurrences	Array of Strings	An array of symbols in a Relation. Each item is adorned and parsed as *currentSymbolString*. A Relation with the constructor "[1][0](1 - [3])[0]" would cause this variable to take the form ["[1", "[0", "(1 - [3", "[3"] (once the final element has been popped). Each item of this array is asynchronously parsed as *currentSymbolString*.
symbolToReplace	String	The term that is to be replaced by *replacementArgument* such that the *Relation* has the parameter *argumentNumber* replaced with the value *currentArgument*. Repeating this process will finish the routine *insertArguments*.
uneliminatedBrackets	String	The result of recursively replacing all bracket pairs with an empty string in listOfBrackets. If the value of this variable is "", then *isBalanced* is set to True and returned, for there are no brackets which are not able to be eliminated by legal traversal rules.

4. CORE FUNCTIONALITY DOCUMENTATION

4.0. NOTES

This section aims to explain what particular functions and methods aim to achieve, and how they achieve it. Oftentimes, the means of achieving an output or side-effect is trivial, so the explanation can be achieved internally within the source code, and thus, many functions do not warrant process descriptions due to their simple nature, but they should be entirely understandable otherwise. The data types can be verified with the data dictionary in the previous section.

4.1. FACT_DATABASE

__init__		
Input	Process	Output / Result
None.	See source.	An empty object.
Description / Purpose: Initialise the database.		

appendFact		
Input	Process	Output / Result
- *fact*: The fact that is to be appended.	See source.	A fact is appended to the database. The caller is passed True if the appendage was successful, and False if the appendage failed.
Description / Purpose: Append a fact to the database.		

deleteFact		
Input	Process	Output / Result
- *factNumber*: The index of the fact that is to be deleted (in terms of the member variable *factList*).	See source.	A fact is removed from the database. The caller is passed True if the deletion was successful, and False if the deletion failed.
Description / Purpose: Delete a fact from the database.		

getFactAggregation		
Input	Process	Output / Result
None.	For every fact in the internal database of facts, save an optimised copy into *optimisedFactList*. Iterate over each fact, and in each iteration, add the current fact multiplied by the product of all prior facts to an aggregate, then, return the aggregate.	A fact representing the system wholly.
Description / Purpose: Create a representation of all the information within the database.		

getFactList		
Input	Process	Output / Result
None.	See source.	List of facts within the object.
Description / Purpose: Read the list of facts that are in the database.		

4.2. FRACTION

__init__		
Input	Process	Output / Result
- *numerator*: The numerator of the fraction. - *denominator*: The value of the fraction's denominator.	See source.	A new object with a defined numerator and denominator.
Description / Purpose: Initialise a fraction which stores two Relations.		

getDenominator		
Input	Process	Output / Result
None.	See source.	The denominator stored in the object.
Description / Purpose: Retrieve the denominator of the Fraction.		

getNumerator		
Input	Process	Output / Result
None.	See source.	The numerator stored in the object.
Description / Purpose: Retrieve the numerator of the Fraction.		

4.3. INTERFACE

__init__		
Input	Process	Output / Result
- *errorOutputMethod*: The method through which any individual error will be parsed as a first argument.	See source.	A new interface object with an empty database, having instructions on what to do with errors.
Description / Purpose: Construct the interface through which the user can operate the P.L.E.		

addFact		
Input	Process	Output / Result
- *fact*: The fact that is to be added.	See source.	A fact is appended to the internal database, or an error is raised indicating that the append failed.
Description / Purpose: Add a fact to the internal database.		

deleteFact		
Input	Process	Output / Result
- *factNumber*: The index of the fact that is to be deleted, in terms of the member *factDatabase's* field *factList*.	See source.	A fact is deleted from the internal database, or an error is raised indicating that the deletion could not be successfully performed.
Description / Purpose: Delete a fact from the internal database.		

generateProcessor		
Input	Process	Output / Result
None.	See source.	A handler to an operationally expensive processor which, when called, allows *retreiveResponse* to be used.
Description / Purpose: Generate a processor who is responsible for the workload in the generation of a response to a user-specified query.		

getFactList		
Input	Process	Output / Result
None.	See source.	List of facts within the database.
Description / Purpose: Retrieve the list of facts that are currently stored in the system.		

getRequest		
Input	Process	Output / Result
None.	See source.	The request that the user most recently set through *setRequest* (otherwise, an error is raised).
Description / Purpose: Recall the request that is currently set in the system (that specified by *setRequest*, having a bearing on the result of *generateProcessor*).		

retrieveReponse		
Input	Process	Output / Result
None.	See source.	The response to a query, E' / (E' - E), or an error message if either processing has not finished or no processor has been generated.
Description / Purpose: Collect the response to the request specified in *setRequest* -- the end goal of the system.		

setRequest		
Input	Process	Output / Result
- *request*: The *Relation* that is sought by the user; the end goal of the system.	See source.	The system is configured to find the solution to the given request once processing is set to begin.
Description / Purpose: Configure the request with which the information of the system is to be developed with respect to; defines the quaesitum of the program wholly.		

4.4. PROCESSOR

__init__		
Input	Process	Output / Result
- *aggregate*: The combined knowledge of the system. - *soughtDevelopment*: The user's request. - *quantumCounter*: A quantum counter configured for *soughtDevelopment*. - *numIterations*: the number of possible states of *quantumCounter*.	See source.	A *Processor* instance with a fully-defined goal.
Description / Purpose: Construct a processor, configuring the end to which the processing will work.		

getProductWith0		
Input	Process	Output / Result
None.	See source.	E' -- product of coefficients in the full development whose respective coefficients in the symbols development are equal to zero.
Description / Purpose: Retrieve half of the result of processing -- half of the entire response of the system.		

getProductWith1		
Input	Process	Output / Result
None.	See source.	E. -- product of coefficients in the full development whose respective coefficients in the symbols development are equal to one.
Description / Purpose: Retrieve half of the result of processing -- half of the entire response of the system		

go		
Input	Process	Output / Result
None.	While processing has not occurred for *numIterations (of __init__)*, create a copy of *aggregate* and *soughtDevelopment*, then insert the arguments in *quantumCounter* (which is also iteratively incremented) -- this way, each possible *quantumCounter* state will be iterated over. Multiply the copy of *aggregate* into the two piles *productWith0* and *productWith1* accordingly with the result of evaluating the copy of *soughtDevelopment*.	Result of processing becomes collectable through the methods *getProductWIth0* and *getProductWith1*.
Description / Purpose: Begin processing.		

hasFinished		
Input	Process	Output / Result
None.	See source.	Whether or not the *go* method has finished execution -- if the results of processing can be read via *getProductWith0* or *getProductWith1*.
Description / Purpose: Determines whether or not the *go* method has been called and finished, namely, whether there are readable results in the repositories *getProductWith1* and *getProductWith0*.		

4.5. QUANTUM_COUNTER

__init__		
Input	Process	Output / Result
- *superPositionStates*: List of bit states for the counter, containing values of 0, 1, or otherwise.	Iterate over each bit in *superPositionStates*. According to the values of each index in this array, append either the value 0, 1, or -1 into the member variable *bitStates*: values of 0 correlate to 0 in *bitStates*, 1 to 1, and anything else to -1.	A *QuantumCounter* object with bits in the states specified by the argument.
Description / Purpose: Initialise the counter in a way such that some bit positions are ignored in incrementation.		

add		
Input	Process	Output / Result
- *numIterations*: The number of incrementations that are to be simulated.	Loop *numIterations* times, iterating each time.	The bit state becomes incremented *numIterations* times.
Description / Purpose: Simulate several calls to the *increment* function -- add some amount of incrementations to the counter.		

copyFrom		
Input	Process	Output / Result
source: The source *QuantumCounter* in the copy.	See source file.	The object is made to reflect the properties of the source.
Description / Purpose: Copy the data from one object into another.		

createCoefficientMap		
Input	Process	Output / Result
- *involvedSymbolList*: List of symbols that are sought to be made as parameters, namely, the list of symbols that are not in quantum states.	Set *largestSymbol* to the largest symbol in *involvedSymbolList*. Set up an array of quantum states (-1) of this length, named *coefficientArray*. For every symbol in the argument *involvedSymbolList*, snap the respective quantum bits in of *coefficientArray* to 0.	The object is overridden with new values such that the counter can be used to increment over the parameters specified in*involvedSymbolList*.
Description / Purpose: Initialise the counter given a list of involved symbols, allowing *QuantumCounter*s to be generated using the more readily available list of parameters in a *Relation*.		

increment		
Input	Process	Output / Result
None.	Set *carry* to 1. Iterate over each bit, right to left, so long as *carry* is still set to 1. Invert every bit state. If the bit state inversion results in a 1 (meaning it was initially 0), set carry to 0, thus terminating the loop (but still performing the inversion).	The internal bit map gets incremented.
Description / Purpose: Add 1 to the right-most bit of the *QuantumCounter*, thus simulating binary addition. Allows the iteration of the counter.		

read		
Input	Process	Output / Result
None.	See source.	The internal map of bit states.
Description / Purpose: Get the current state of the counter as an array.		

toString		
Input	Process	Output / Result
None.	Iterate over each bit in the member variable *bitStates*. If the bit is not in a quantum state, append representative data to the string accordingly (obvious in source).	Stringified representation of the bit map.
Description / Purpose: Convert the object into a form that can be printed to the screen.		

4.6. RELATION

__add__		
Input	Process	Output / Result
- *arg*: The second operand to the addition operation.	See source.	*Relation* which is the sum of the two arguments.
Description / Purpose: Allows the addition of two *Relations*, simulating an OR Boolean operation (noting that the two operands must be mutually exclusive).		

__eq__		
Input	Process	Output / Result
- *arg*: The second operand to the equality comparator.	Combine a list of symbols in both *arg* and *this* (local member), and create a *QuantumCounter* named *coefficientMap* using this array. Next, iterate over every possible state of *coefficientMap* by noting that the number of states is exponentially proportional (in base 2) to the number of symbols in the aggregate list of symbols. In each iteration, evaluate a copy of both *Relations* with *coefficientMap*, and evaluating each result. If the return value of evaluation for each of the two *Relations* are ever not equal, set the flag *hasEncounteredDiscrepancy* to True, and return accordingly.	Whether or not the two expressions are equal.
Description / Purpose: Determine whether two *Relations* are functionally equivalent.		

__init__		
Input	Process	Output / Result
- *constructor*: The *Relation* in string form (see manual).	See source.	An object reflecting the data in the constructor.
Description / Purpose: Initialise a *Relation* -- the fundamental component of information in the P.L.E.		

__mul__		
Input	Process	Output / Result
- *arg*: Second operand to the multiplication operation.	See source.	*Relation* which is the multiple of the two operands.
Description / Purpose: Simulate the Boolean operation AND, which is required for information manipulation.		

__ne__		
Input	Process	Output / Result
- *arg*: The second operand in the standard non-equivalence comparator.	See source.	Whether or not the two expressions are not equal.
Description / Purpose: The opposite of the *__eq__* function. Used for shorthand reference.		

__sub__		
Input	Process	Output / Result
- *arg*: The second operand to the subtraction operation.	See source.	*Relation* that is the result of a subtraction operation on the two arguments.
Description / Purpose: Combined with other operations, allows the simulation of the binary operation NOT, necessary for information manipulation.		

copyFrom		
Input	Process	Output / Result
- *source*: The source *Relation* from which data is to be copied.	See source.	The object is made to reflect the properties of the source.
Description / Purpose: Copy the data from another *Relation* into the current object.		

eliminateSymbol		
Input	Process	Output / Result
- *symbolNumber*: The identifier of the symbol that is to be eliminate (set to a quantum state).	Create a *QuantumCounter* object *coefficientMap* with a parameter according to the method's argument. Evaluate two copies of the *Relation's* data respectively with the raw *coefficientMap*, and with the same argument after one incrementation. Multiply the results, and copy it locally.	The object state changes to the result of eliminating the symbol symbolNumber.
Description / Purpose: Eliminate a symbol in a *Relation*, simulating what state the information would be in were the eliminated symbol (*symbolNumber*) in a quantum superposition.		

evaluate		
Input	Process	Output / Result
None.	See source.	The result of evaluating the data within the object.
Description / Purpose: Evaluate a *Relation* which has no unfilled parameters. Allows the coefficients of constituents to be solved for.		

getSymbolList		
Input	Process	Output / Result
None.	Split the data string at each closing symbol bracket, then remove the final element of the array. Iterate over each element, splitting them individually at an open bracket and reading to the right, such that what was contained within each bracket can now be read. Implicitly convert this value into an integer, and append it to *symbolList* if it is not already in the array.	A list of symbols (parameters) that are involved in the object.
Description / Purpose: Get a list of symbols that are involved in the *Relation*, namely, the parameters of the mathematical function.		

insertArguments		
Input	Process	Output / Result
- *quantumCounter*: Map of arguments that are to be inserted.	Get a list of parameters in *quantumCounter*, and iterate over each one, performing a search and replace in the member string *data* if the argument is not in a quantum state.	The data in the object is changed so that any non-quantum bits in *quantumCounter* now hold either the values 0 or 1, as specified by the argument.
Description / Purpose: Insert arguments into the *Relation* so as to allow either evaluation to produce a Boolean value, or to create a coefficient in the form of a *Relation*.		

isValid		
Input	Process	Output / Result
None.	See source.	Whether or not the data stored within the object is valid (particularly whether it is safe to evaluate).
Description / Purpose: Determine whether or not a *Relation* is malformed by the set of rules in the manual. Malformed *Relations* are dangerous.		

optimise		
Input	Process	Output / Result
None.	See source.	Functionally, there is no change.
Description / Purpose: Attempt to shrink the *Relation* (in terms of memory) while also keeping it functionally equivalent to its previous form.		

5. CORE SOURCE CODE

5.0. NOTES

Subsequent maintenance may render some of the files under this heading as being slightly out dated. Any modifications are however documented in maintenance logs through section `VI.. Under this heading and the next two, it must also be noted that some source files are missing, such as __init__.py. This is because such files do not provide actual functionality, but just the means through which other files operate.

5.1. FACT_DATABASE.PY

```
# FactDatabase/FactDatabase.py
# Nicholas Killeen,
# 19th June 2016.
# Class to store and perform operations on a collection of facts.

import Relation.Relation as Relation;

# Data structure to store and manipulate a group of facts.
class FactDatabase:
    def __init__(this):
        this.factList = [];

    def appendFact(this, fact):
        if (hasattr(fact, "isValid")):
            if (fact.isValid()):
                this.factList.append(fact);
                successState = True;
            else:
                successState = False;
                # the Relation is malformed
        else:
            successState = False;
            # the object is not of type Relation
        return successState;

    def deleteFact(this, factNumber):
        if (factNumber >= 0 and factNumber < len(this.factList)):
            this.factList.pop(factNumber);
            successState = True;
        else:
            successState = False;
        return successState;

    # Returns one Relation containing the combined knowledge of all
    # facts in the system.
    def getFactAggregation(this):
        optimisedFactList = [];
        for fact in this.factList:
            optimisedFact = Relation.Relation();
```

```
            optimisedFact.copyFrom(fact);
            optimisedFact.optimise();
            optimisedFactList.append(optimisedFact);

        UNITY = Relation.Relation('1');
        aggregate = Relation.Relation('0');
        factNumber = 0;
        numFacts = len(optimisedFactList);
        while (factNumber < numFacts):
            priorFactNumber = 0;
            productOfPriorFacts = Relation.Relation('1');
            while (priorFactNumber < factNumber):
                productOfPriorFacts = productOfPriorFacts * (UNITY -
                    optimisedFactList[priorFactNumber]);
                priorFactNumber += 1;
            currentFact = optimisedFactList[factNumber];
            aggregate += currentFact * productOfPriorFacts;
            factNumber += 1;
        aggregate.optimise();
        return aggregate;

    def getFactList(this):
        return this.factList;
```

5.2. FRACTION.PY

```
# Fraction/Fraction.py
# Nicholas Killeen,
# 19th June 2016.
# Container class to store fractions involving Relations.

# Stores a numerator and a denominator.
class Fraction:
    def __init__(this, numerator, denominator):
        this.numerator = numerator;
        this.denominator = denominator;

    def getDenominator(this):
        return this.denominator;

    def getNumerator(this):
        return this.numerator;
```

5.3. INTERFACE.PY

```
# Interface/Interface.py
# Nicholas Killeen,
# 19th June 2016.
# Interfacing class for a Propositional Logic Engine, allowing facts to
# be entered in mathematical form, and queries to be asked and answered.

import FactDatabase.FactDatabase as FactDatabase;
import Relation.Relation as Relation;
```

```
import QuantumCounter.QuantumCounter as QuantumCounter;
import Processor.Processor as Processor;
import Fraction.Fraction as Fraction;

# Class defining methods of interaction between a caller and an internal
# instance of FactDatabase.
class Interface:
    def __init__(this, errorOutputMethod = None):
        if (errorOutputMethod == None):
            errorOutputMethod = lambda errorMessage: None;
            # by default, errors are suppressed
        this.errorOutputMethod = errorOutputMethod;
        this.factDatabase = FactDatabase.FactDatabase();
        this.isRequestValid = False;
        this.processor = None;
        this.request = None;

    def addFact(this, fact):
        successState = this.factDatabase.appendFact(fact);
        if (successState == False):
            this.errorOutputMethod("Failed to append fact: fact is "
                "malformed.");

    def deleteFact(this, factNumber):
        successState = this.factDatabase.deleteFact(factNumber);
        if (successState == False):
            this.errorOutputMethod("Failed to delete fact: fact is out "
                "of range");

    # Returns a handle to the processor, through which the process can
    # be enacted.
    def generateProcessor(this):
        handler = None;
        if (this.isRequestValid):
            aggregate = this.factDatabase.getFactAggregation();
            request = this.getRequest();
            requestSymbolList = request.getSymbolList();
            totalWork = 2 ** len(requestSymbolList);
            coefficientMap = QuantumCounter.QuantumCounter();
            coefficientMap.createCoefficientMap(requestSymbolList);
            this.processor = Processor.Processor(aggregate, request,
                coefficientMap, totalWork);
            handler = this.processor.go;
        else:
            this.errorOutputMethod("Failed to declare environment: "
                "request is not defined.");
        return handler;

    def getFactList(this):
        return this.factDatabase.getFactList();

    # Recall the currently set request.
    def getRequest(this):
```

```
        request = None;
        if (this.isRequestValid):
            request = this.request;
        else:
            this.errorOutputMethod("Failed to get request: request is "
                "not defined.");
        return request;

    def retrieveResponse(this):
        returnValue = None;
        if (this.processor == None):
            this.errorOutputMethod("Failed to retrieve fraction: no "
                "processor has been generated.");
        else:
            if (this.processor.hasFinished()):
                returnValue = _aggregateResponse(this.processor);
            else:
                this.errorOutputMethod("Failed to retrieve fraction -- "
                    "the processor is not ready.");
        return returnValue;

    def setRequest(this, request):
        isValid = request.isValid();
        this.isRequestValid = isValid;
        if (isValid):
            this.request = request;
        else:
            this.errorOutputMethod("Failed to set request: request is "
                "malformed");

def _aggregateResponse(repository):
    productWith0 = repository.getProductWith0();
    productWith1 = repository.getProductWith1();
    numerator = productWith0;
    denominator = productWith0 - productWith1;
    return Fraction.Fraction(numerator, denominator);
```

5.4. PROCESSOR.PY

```
# Processor/Processor.py
# Nicholas Killeen,
# 19th June 2016.
# Module to perform mass calculations so as to generate a partial answer
# to a request from the P.L.E.

import Relation.Relation as Relation;
import QuantumCounter.QuantumCounter as QuantumCounter;

# Class to calculate the products of coefficients in a representative
# aggregation of facts with respect to the list of symbols that comprise
# a request, and then, sort these coefficients into two piles.
class Processor:
    def __init__(this, aggregate, soughtDevelopment, quantumCounter,
```

```
        numIterations):
        this.aggregate = aggregate;
        this.finishedProcessing = False;
        this.productWith0 = Relation.Relation("1");
        this.productWith1 = Relation.Relation("1");
        this.quantumCounter = quantumCounter;
        this.remainingIterations = numIterations;
        this.soughtDevelopment = soughtDevelopment;

    def getProductWith0(this):
        return this.productWith0;

    def getProductWith1(this):
        return this.productWith1;

    # Iterates through numIterations coefficient maps, and stores the
    # results locally to be retrieved with getProductWith0 /
    # getProductWith1.
    def go(this):
        while (this.remainingIterations > 0):
            fullDevelopmentCopy = Relation.Relation();
            fullDevelopmentCopy.copyFrom(this.aggregate);
            fullDevelopmentCopy.insertArguments(this.quantumCounter);
            soughtDevelopmentCopy = Relation.Relation();
            soughtDevelopmentCopy.copyFrom(this.soughtDevelopment);
            soughtDevelopmentCopy.insertArguments(this.quantumCounter);

            coefficient = soughtDevelopmentCopy.evaluate();
            if (coefficient == 0):
                this.productWith0 = (this.productWith0 *
                    fullDevelopmentCopy);
            else:
                this.productWith1 = (this.productWith1 *
                    fullDevelopmentCopy);

            this.quantumCounter.increment();
            this.remainingIterations -= 1;
        this.finishedProcessing = True;

    def hasFinished(this):
        return this.finishedProcessing;
```

5.5. QUANTUM_COUNTER.PY

```
# QuantumCounter/QuantumCounter.py
# Nicholas Killeen,
# 19th June 2016.
# Object definition for a binary counter, supporting rudimentary quantum
# states.

# Class to count in binary, but skipping quantum bit states.
class QuantumCounter:
    def __init__(this, superPositionStates = []):
```

```
        this.bitStates = [];
        bitNumber = 0;
        numBits = len(superPositionStates);
        while (bitNumber < numBits):
            if (superPositionStates[bitNumber] == 0):
                this.bitStates.append(0);
            elif (superPositionStates[bitNumber] == 1):
                this.bitStates.append(1);
            else:
                this.bitStates.append(-1);
                # -1 is used to represent a quantum superposition,
                # namely, something other than 0 or 1
            bitNumber += 1;

    # Simulate numIterations iterations.
    def add(this, numIterations):
        iteration = 0;
        while (iteration < numIterations):
            this.increment()
            iteration += 1;

    def copyFrom(this, source):
        this.bitStates = source.bitStates;

    # Set up a QuantumCounter, where the symbols in involvedSymbolList
    # take the value of 0, and all other symbols are in superposition.
    def createCoefficientMap(this, involvedSymbolList):
        largestSymbol = 0;
        if (len(involvedSymbolList) > 0):
            largestSymbol = (max(involvedSymbolList));
        coefficientArray = [-1] * (largestSymbol + 1);
        for symbol in involvedSymbolList:
            coefficientArray[symbol] = 0;
        coefficientMap = QuantumCounter(coefficientArray);
        this.copyFrom(coefficientMap);

    # Increment the current state of the counter, moving over quantum
    # bits. Overflow resets the counter.
    def increment(this):
        bitNumber = len(this.bitStates) - 1;
        carry = 1;
        # iterate over each bit, right to left, stop if there is no
        # longer a bit to carry
        while (bitNumber >= 0 and carry == 1):
            currentState = this.bitStates[bitNumber];
            if (currentState == 0):
                this.bitStates[bitNumber] = 1;
                carry = 0;
            elif (currentState == 1):
                this.bitStates[bitNumber] = 0;
            bitNumber -= 1;

    def read(this):
```

```
        return this.bitStates;

    def toString(this):
        string = "";
        bitNumber = 0;
        numBits = len(this.bitStates)
        while (bitNumber < numBits):
            currentBit = this.bitStates[bitNumber];
            if (currentBit == 0):
                string += "(1 - [{0}])".format(bitNumber);
            elif (currentBit == 1):
                string += "[{0}]".format(bitNumber);
            bitNumber += 1;
        return string;
```

5.6. RELATION.PY

```
# Relation/Relation.py
# Nicholas Killeen,
# 19th June 2016.
# Class to store and algebraically manipulate Boolean relations.

import QuantumCounter.QuantumCounter as QuantumCounter;

# Stores and manipulates a connection of Boolean symbols.
# Unsafe unless Relation::isValid(this) -> True.
class Relation:
    def __add__(this, arg):
        return Relation(this.data + "+" + arg.data);

    def __eq__(this, arg):
        # compile a list of all symbols in the two Relations, then use
        # that list to construct a QuantumCounter as a coefficientMap
        symbolList = this.getSymbolList();
        symbolList.extend(arg.getSymbolList());
        symbolList = list(set(symbolList)); # remove any repeated items
        coefficientMap = QuantumCounter.QuantumCounter();
        coefficientMap.createCoefficientMap(symbolList);

        # iterate through every possible state of coefficients, if pairs
        # do not match from both relations, they are not equal
        numInvolvedSymbols = len(symbolList);
        numCoefficients = 2 ** numInvolvedSymbols;
        coefficientNumber = 0;
        hasEncounteredDiscrepancy = False;
        while (coefficientNumber < numCoefficients):
            relationA = Relation();
            relationA.copyFrom(this);
            relationA.insertArguments(coefficientMap);
            relationB = Relation();
            relationB.copyFrom(arg);
            relationB.insertArguments(coefficientMap);
            if (relationA.evaluate() != relationB.evaluate()):
```

```
                # not all non-zero coefficients are treated equally
                hasEncounteredDiscrepancy = True;
            coefficientMap.increment();
            coefficientNumber += 1;
        return not hasEncounteredDiscrepancy;

    def __init__(this, constructor = ""):
        this.data = constructor;

    def __mul__(this, arg):
        return Relation("(" + this.data + ")(" + arg.data + ")");

    def __ne__(this, arg):
        return not this == arg;

    def __sub__(this, arg):
        return Relation(this.data + "-(" + arg.data + ")");

    def copyFrom(this, source):
        this.data = source.data;

    def eliminateSymbol(this, symbolNumber):
        coefficientMap = QuantumCounter.QuantumCounter();
        coefficientMap.createCoefficientMap([symbolNumber]);

        relationWith0 = Relation();
        relationWith0.copyFrom(this);
        relationWith0.insertArguments(coefficientMap);

        coefficientMap.increment();
        relationWith1 = Relation();
        relationWith1.copyFrom(this);
        relationWith1.insertArguments(coefficientMap);

        resultOfElimination = relationWith0 * relationWith1;
        this.copyFrom(resultOfElimination);

    # Evaluates a Relation that contains no symbols.
    # Unsafe method; must not be called unless:
    #  - Relation::isValid(this) -> True.
    #  - Relation::getSymbolList(this) -> [].
    def evaluate(this):
        evaluableString = this.data.replace(")(", ")*(");
        return eval(evaluableString);

    def getSymbolList(this):
        symbolList = [];
        symbolOccurrences = this.data.split(']');
        symbolOccurrences.pop();
        # the final element of symbolOccurrences is garbage, containing
        # the rest of the data string after the final symbol
        numOccurrences = len(symbolOccurrences);
        occurrence = 0;
```

```
        while (occurrence < numOccurrences):
            currentSymbolString = symbolOccurrences[occurrence];
            currentSymbolString = currentSymbolString.split('[')[1];
            currentSymbol = int(currentSymbolString);
            if (symbolList.count(currentSymbol) == 0):
                symbolList.append(currentSymbol);
            occurrence += 1;
        return symbolList;

    def insertArguments(this, quantumCounter):
        argumentMap = quantumCounter.read();
        argumentNumber = 0;
        numArguments = len(argumentMap);
        while (argumentNumber < numArguments):
            currentArgument = argumentMap[argumentNumber];
            if (currentArgument == 0 or currentArgument == 1):
                symbolToReplace = '[' + str(argumentNumber) + ']';
                replacementArgument = '(' + str(currentArgument) + ')';
                this.data = this.data.replace(symbolToReplace,
                    replacementArgument);
            argumentNumber += 1;

    # Determines whether a Relation is invalid. Malformed Relations are
    # potentially unsafe.
    def isValid(this):
        dataString = this.data;
        isValid = _areAllCharactersLegal(dataString);
        if (isValid):
            isValid = _areCombinationsLegal(dataString);
            # check for disallowed character adjacencies
        if (isValid):
            isValid = _isNestingLegal(dataString);
            # check that each bracket occurs in a legal pair
        if (isValid):
            isValid = _areNumbersLegal(dataString);
            # check that no numbers have trailing 0s
        if (isValid):
            isValid = (this == this * this);
            # check that the expression exists in the Boolean domain,
            # {0, 1}, as by the condition n = n^2
        return isValid;

    # Attempt to optimise the data a Relation instance stores.
    # Should only impact performance.
    def optimise(this):
        this.data = this.data.replace(' ', "");
        this.data = this.data.replace(")(1)(", ")(");
        this.data = this.data.replace(")(1)", "))");


def _areAllCharactersLegal(dataString):
    hasFoundIllegalChar = False;
    for character in dataString:
```

```
        if ("0123456789 +-()[]".count(character) == 0):
            hasFoundIllegalChar = True;
    return not hasFoundIllegalChar;

def _areCombinationsLegal(dataString):
    reducedData = dataString;

    # turn all numberical expressions into 'n'
    for digit in "0123456789":
        reducedData = _recursivelyReplace(reducedData, digit, 'n');
    reducedData = _recursivelyReplace(reducedData, "nn", 'n');

    # collapse all symbols "[n]", since they are equivalent to "(n)"
    reducedData = _recursivelyReplace(reducedData, "[n]", "(n)");

    # replace all multipliable terms 'n' with 'M'
    reducedData = _recursivelyReplace(reducedData, '(n)', "(M)");

    # replace all non-multiplable terms 'n' with "(B)"
    reducedData = _recursivelyReplace(reducedData, 'n', "(B)");

    # replace all '-' signs with '+', since they are of the same nature
    reducedData = _recursivelyReplace(reducedData, "-", "+");

    # remove all double whitespace as they are treated equally to single
    # spaces
    reducedData = _recursivelyReplace(reducedData, "  ", ' ');

    # flag the beginning and end of the string with "O", and then remove
    # meaningless whitespace at the start and end of the string
    reducedData = 'O' + reducedData + 'O';
    reducedData = reducedData.replace("O ", 'O');
    reducedData = reducedData.replace(" O", 'O');

    # search the reduced string for all possible illegal expressions
    LIST_OF_ILLEGAL_EXPRESSIONS = ["M)(B", "B)(M" , ") (", "()", "( )",
        "++", "+ +", "+)", "(+", "[", "]", "OM", "OB", "O)", "O+", "+O",
        "(O", "BO", "MO", "OO", "B)(", ")(B"];
    hasFoundIllegalCombination = False;
    for expression in LIST_OF_ILLEGAL_EXPRESSIONS:
        if (reducedData.count(expression) != 0):
            hasFoundIllegalCombination = True;

    return not hasFoundIllegalCombination;

def _areNumbersLegal(dataString):
    # compile a list of all numbers
    involvedNumbers = [""];
    currentNumber = 0;
    for character in dataString:
        if ("0123456789".count(character) != 0):
            involvedNumbers[currentNumber] += character;
        elif involvedNumbers[currentNumber] != "":
```

```
            involvedNumbers.append("");
            currentNumber += 1;

    # check that each number has no leading zeros if it is more than
    # one digit long
    hasFoundIllegalNum = False;
    for number in involvedNumbers:
        if (len(number) > 1):
            if (number[0] == '0'):
                hasFoundIllegalNum = True;

    return not hasFoundIllegalNum;

def _isNestingLegal(dataString):
    # generate a list of all brackets
    listOfBrackets = "";
    for character in dataString:
        if (character == ')' or character =='('):
            listOfBrackets += character;
        elif (character == ']'):
            listOfBrackets += ')';
        elif (character == '['):
            listOfBrackets += '(';

    # eliminate all legal bracket pairs
    uneliminatedBrackets = _recursivelyReplace(listOfBrackets, "()", "");
    if (uneliminatedBrackets == ""):
        isBalanced = True;
    else:
        isBalanced = False;
    return isBalanced;

def _recursivelyReplace(string, searchTerm, replacementTerm):
    if (replacementTerm.count(searchTerm) != 0):
        assert(searchTerm == replacementTerm or string == "");
    stringBeforeChange = "";
    stringAfterChange = string;
    while (stringAfterChange != stringBeforeChange):
        stringBeforeChange = stringAfterChange;
        stringAfterChange = stringBeforeChange.replace(searchTerm,
            replacementTerm);
    return stringAfterChange;
```

6. CLI SOURCE CODE

6.1. RUN_C_L_I.PY

```
# runCLI.py
# Nicholas Killeen,
# 27th June 2016.
# Runs a command line interface for the Propositional Logic Engine,
# with limited functionality.

import FactDatabase.FactDatabase as FactDatabase;
import Fraction.Fraction as Fraction;
import Interface.Interface as Interface;
import Processor.Processor as Processor;
import QuantumCounter.QuantumCounter as QuantumCounter;
import Relation.Relation as Relation;

def runCLI():
    interface = Interface.Interface(print);
    userWishesToExit = False;
    print("Running PLE, welcome!");
    print("Press the return key twice to quit.");
    while (not userWishesToExit):
        command = input("\n1. Add a fact"
            "\n2. Delete a fact"
            "\n3. Show a list of facts "
            "\n4. Request a solution.\n>>> ");
        if (command == "1"):
            # add
            constructor = input("Enter the fact to add:\n>>> ");
            constructor = _transpose(constructor);
            fact = Relation.Relation(constructor);
            interface.addFact(fact);
        elif (command == "2"):
            # delete
            rawInput = input("Enter the fact number to delete:\n>>> ");
            try:
                factNumberToDelete = int(rawInput) - 1;
                interface.deleteFact(factNumberToDelete);
            except Exception as error:
                print("Could not delete fact.");
        elif (command == "3"):
            # show list (in transposed form)
            factList = interface.getFactList();
            factNumber = 0;
            numFacts = len(factList);
            while (factNumber < numFacts):
                currentFactStr = factList[factNumber].data;
                print("{0} -> {1}".format(factNumber + 1,
                    currentFactStr));
                factNumber += 1;
            if (numFacts == 0):
```

```
                print("There are no facts.");
        elif (command == "4"):
            # request solution
            request = Relation.Relation(input("Enter the request:\n>>> "
                ));
            if (request.isValid()):
                interface.setRequest(request);
                handler = interface.generateProcessor();
                handler();
                response = interface.retrieveResponse();
                _formatSoughtInformation(response, print);
            else:
                print("The request is malformed.");
        elif (command == ""):
            userWishesToExit = True;
        else:
            print("Did not recognise command.");

# Given a fraction, formats it as a development of constituents, whose
# respective coefficients can take the values 0, 1, 0/0, or 1/0.
def _formatSoughtInformation(fraction, outputMethod):
    numerator = fraction.getNumerator();
    denominator = fraction.getDenominator();
    involvedSymbolList = denominator.getSymbolList();
    coefficientMap = QuantumCounter.QuantumCounter();
    coefficientMap.createCoefficientMap(involvedSymbolList);

    coefficientNumber = 0;
    numCoefficients = 2 ** len(involvedSymbolList);
    while (coefficientNumber < numCoefficients):

        numeratorCopy = Relation.Relation();
        numeratorCopy.copyFrom(numerator);
        numeratorCopy.insertArguments(coefficientMap);
        currentNumerator = numeratorCopy.evaluate();

        denominatorCopy = Relation.Relation();
        denominatorCopy.copyFrom(denominator);
        denominatorCopy.insertArguments(coefficientMap);
        currentDenominator = denominatorCopy.evaluate();

        if (currentNumerator == 0 and currentDenominator == 0):
            outputMethod("0/0{0}".format(coefficientMap.toString()));
        elif (currentNumerator == 1 and currentDenominator == 1):
            outputMethod("1{0}".format(coefficientMap.toString()));
        elif (currentNumerator == 1 and currentDenominator == 0):
            outputMethod("1/0{0}".format(coefficientMap.toString()));
        # else the coefficient is naught (0/1), and does not need to
        # be printed

        coefficientMap.increment();
        coefficientNumber += 1;
```

```
# Function to convert Relations in the form "subject copula predicate"
# ("A = B") -- which the user understands easily -- to the more machine
# understandable form "A(1 - B) + B(1 - A) = 0".
def _transpose(constructor):
    transposedConstructor = None;
    slicedConstructor = constructor.split('=');
    if (len(slicedConstructor) == 2):
        subject = slicedConstructor[0];
        predicate = slicedConstructor[1];
        transposedConstructor = ("({0})(1 - ({1})) + ({1})(1 - ({0}))"
            "").format(subject, predicate);
    else:
        # set the constructor to something illegal
        transposedConstructor = 'v';
    return transposedConstructor;
```

7. GUI SOURCE CODE

7.1. _G_U_I.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="application-name" content="Propositional Logic Engine"
          />
    <meta name="author" content="Nicholas Killeen" />
    <meta name="date" content="31st July 2016" />
    <meta name="release-version" content="1.0" />
    <meta name="description" content="Program to perform algebraic
          manipulations on facts in order to reach a requested
          solution." />
    <title>Propositional Logic Engine</title>
    <style>
        /* wildcards */
        *[hiddenDiv="true"]
        {
            display: none;
        }

        /* tag rules */
        body
        {
            background-color: #ECECEC;
            font-family: sans-serif;
            height: 100%;
            margin: 0;
            min-height: 500px;
            min-width: 700px;
            padding: 0;
            width: 100%;
        }
        html
        {
            -webkit-user-drag: none;
            -webkit-user-select: none;
            cursor: default;
            height: 100%;
            width: 100%;
        }

        /* class rules */
        .button
        {
            cursor: pointer;
        }
        .button:active
        {
```

```
            background-color: #F7F7F7;
        }
        .button:hover:not(:active)
        {
            background-color: #ECECEC;
        }
        .button.left
        {
            border-left: 1px solid #DADADA;
            border-right: 1px solid #DADADA;
            float: left;
            padding-left: 5px;
            padding-right: 5px;
        }
        .button.middle
        {
            border-right: 1px solid #DADADA;
            float: left;
            padding-left: 5px;
            padding-right: 5px;
        }
        .button.right
        {
            float: left;
            padding-right: 5px;
            padding-left: 5px;
        }
        .buttonHolder
        {
            float: right;
        }
        .constituent:hover
        {
            background-color: #F7F7F7;
        }
        .content
        {
            background-color: #FFF;
            border: 1px solid #DADADA;
            padding: 5px;
        }
        .contentComponent.upper
        {
            border-bottom: 1px solid #DADADA;
            padding-bottom: 5px;
        }
        .contentComponent.middle
        {
            border-bottom: 1px solid #DADADA;
            padding-bottom: 5px;
            padding-top: 5px;
        }
        .contentComponent.lower
```

```
        {
            padding-top: 5px;
        }
        .description
        {
            height: 100px;
            overflow-y: scroll; /* auto */
        }
        .determinateConstituent
        {
            padding: 5px;
        }
        .determinateConstituent:not(:last-child)
        {
            border-bottom: 1px solid #ECECEC;
        }
        .editable
        {
            word-wrap: break-word;
        }
        .editable[editable="true"]
        {
            -webkit-user-modify: read-write-plaintext-only;
            -webkit-user-select: initial;
            cursor: text;
        }
        .editable:focus
        {
            outline: none;
        }
        .fact
        {
            border-bottom: 1px solid #ECECEC;
            padding: 5px;
        }
        .fact, .symbolName
        {
            -webkit-box-sizing: border-box;
            -webkit-user-modify: read-write-plaintext-only;
            -webkit-user-select: initial;
            cursor: text;
            width: 100%;
            word-wrap: break-word;
        }
        .fact:hover, .symbol:hover
        {
            background-color: #F7F7F7;
        }
        .fact:focus, .symbolName:focus
        {
            outline: none;
        }
        .fact[selected="true"], .symbol[selected="true"],
```

```
            .constituent[selected="true"]
        {
            background-color: #ECECEC;
        }
        .fact[valid="true"]
        {
            border-right: 5px solid #008000;
        }
        .fact[valid="false"]
        {
            border-right: 5px solid #FF0000;
        }
        .fact[valid="maybe"]
        {
            border-right: 5px solid #FF8000;
        }
        .footSpan
        {
            margin-top: 10px;
        }
        .heading
        {
            text-align: center;
        }
        .infiniteConstituent
        {
            padding: 5px;
        }
        .infiniteConstituent:not(:last-child)
        {
            border-bottom: 1px solid #ECECEC;
        }
        .involvedSymbol
        {
            border-bottom: 1px solid #ECECEC;
            padding: 5px;
        }
        .involvedSymbolDefinition
        {
            overflow: hidden;
            padding-left: 5px;
            word-wrap: break-word;
        }
        .involvedSymbolNumber
        {
            border-right: 1px solid #ECECEC;
            float: left;
            padding-right: 5px;
            word-wrap: break-word;
        }
        .list
        {
            padding-right: 5px;
```

```
            height: 200px;
            overflow-y: scroll;
        }
        .panelInner.left
        {
            margin-right: 5px;
        }
        .panelInner.offset
        {
            margin-top: 10px;
        }
        .panelInner.right
        {
            margin-left: 5px;
        }
        .panelOuter
        {
            float: left;
            width: 50%;
        }
        .placeholder:empty:before
        {
            color: #808080;
            content: attr(placeholder);
        }
        .placeholder:empty:focus:before
        {
            content: none;
        }
        .quantumConstituent
        {
            padding: 5px;
        }
        .quantumConstituent:not(:last-child)
        {
            border-bottom: 1px solid #ECECEC;
        }
        .symbol /* inherits some properties of .fact */
        {
            border-bottom: 1px solid #ECECEC;
            padding: 5px;
        }
        .symbol[eliminated="true"]
        {
            border-right: 5px solid #A9C6E4;
        }
        .symbol[eliminated="false"]
        {
            border-right: 5px solid #0080FF;
        }
        .symbol[eliminated="true"] > *
        {
            color: #808080;
```

```
        }
        .symbolNumber
        {
            border-right: 1px solid #DADADA;
            float: left;
            padding-right: 5px;
            word-wrap: break-word;
        }
        .symbolNumber:empty:after
        {
            color: #808080;
        }
        .symbolNumber:empty:before
        {
            color: #808080;
            content: "[ ";
        }
        .symbolNumber:after
        {
            content: "]";
        }
        .symbolNumber:before
        {
            content: "[";
        }
        .symbolName /* inherits some properties of .fact */
        {
            overflow: hidden;
            padding-left: 5px;
            width: auto;
        }

        /* ID rules */
        #determinateConstituentContainer
        {
            margin-top: 5px;
            padding-top: 5px;
        }
        #footHolder
        {
            float: left;
            padding-bottom: 10px;
            width: 100%;
        }
        #outBox
        {
            -webkit-user-select: text;
        }
        #outBoxInf
        {
            -webkit-user-select: text;
            margin-top: 10px;
        }
```

```
        #panelHolder
        {
            height: auto;
        }
        #quantumConstituentContainer
        {
            border-bottom: 1px solid #DADADA;
            margin-top: 5px;
        }
        #quaesitumInsertionPoint
        {
            border-right: 1px solid #DADADA;
            overflow-y: auto;
            overflow-x: hidden;
            width: auto;
        }
        #request
        {
            float: right;
            padding-left: 5px;
            padding-right: 5px;
        }
        #tortilla
        {
            padding: 10px;
            width: auto;
        }
    </style>
    <script name="front-end definitions" type="text/javascript">
        function checkValidity(factNode)
        {
            while (factNode.innerText.search("\n") != -1)
            {
                factNode.innerText = factNode.innerText.replace("\n", "");
            }
            var constructor = factNode.innerText;
            while (constructor.search("\n") != -1)
            {
                constructor = constructor.replace("\n", "");
            }
            // this loop appears redundant, but innerText is updated
            // asynchronously, making it necessary

            Python.isValid(constructor);
            isValid = Python.register;
            Python.register = undefined;
            if (isValid == true)
            {
                factNode.setAttribute("valid", "true");
            }
            else if (isValid == false)
            {
                factNode.setAttribute("valid", "false");
```

```
            }
            else
            {
                // there were too many symbols in the fact, so checking
                // would take too long
                factNode.setAttribute("valid", "maybe");
            }
        }

        function deselectSelectedNodes()
        {
            // deselect the current constituent
            var constituentNodes =
                document.getElementsByClassName("constituent");
            for (var constituent = 0; constituent <
                constituentNodes.length; ++constituent)
            {
                if (constituentNodes[constituent].getAttribute(
                    "selected") == "true")
                {
                    constituentNodes[constituent].setAttribute(
                        "selected", "false");
                }
            }
            // deselect current fact
            var factNodes = document.getElementsByClassName("fact");
            for (var fact = 0; fact < factNodes.length; ++fact)
            {
                if (factNodes[fact].getAttribute("selected") == "true")
                {
                    factNodes[fact].setAttribute("selected", "false");
                }
            }
            document.getElementById("symbolMap").innerHTML = "";
            document.getElementById("factDescription").innerText = "";
            document.getElementById("factDescription").setAttribute(
                "editable", "false");
            document.getElementById("factDescription").setAttribute(
                "placeholder", "");
        }

        function getElimiatedSymbols()
        {
            var eliminatedSymbols = [];
            var symbols = document.getElementsByClassName("symbol");
            for (var symbolNumber = 0; symbolNumber < symbols.length;
                ++symbolNumber)
            {
                var currentSymbolNode = symbols[symbolNumber]
                if (currentSymbolNode.getAttribute("eliminated") ==
                    "true")
                {
                    var currentSymbolNumber =
```

```
                    currentSymbolNode.children[0].innerText;
                if (currentSymbolNumber != "")
                {
                    eliminatedSymbols[eliminatedSymbols.length] =
                        parseInt(currentSymbolNumber);
                }
            }
        }
        return eliminatedSymbols;
    }

    // Stop any illegal values from remaining as a symbol number
    // after events that could potentially change the value.
    function formatSymbolNumber(event)
    {
        if (event.target.classList.contains("symbolNumber"))
        {
            var oldRange = getSelection().getRangeAt(0);
            var oldInnerText = event.target.innerText;
            if (oldInnerText.length > 3)
            {
                event.target.innerText = oldInnerText.slice(0, 3);
            }

            var innerText = oldInnerText;
            var newInnerText = "";
            while (innerText.search(/[0-9]/) != -1)
            {
                newInnerText += innerText[innerText.search(
                    /[0-9]/)];
                innerText = innerText.replace(/[0-9]/, "");
            }
            if (oldInnerText.length > 1 && oldInnerText[0] == '0')
            {
                var innerText = newInnerText;
                while (innerText.length > 1 && innerText[0] == '0')
                {
                    innerText = innerText.slice(1);
                }
                event.target.innerText = innerText;
            }
            else if (oldInnerText != newInnerText)
            {
                event.target.innerText = newInnerText;
            }

            // refocus the caret
            var selectionObject = getSelection();
            selectionObject.removeAllRanges();
            selectionObject.addRange(oldRange);
        }
    }
```

```
        // Remove whitespace such that the node is treated as *:empty {}
        // by CSS.
        function removeWhitespace(targetNode)
        {
            if (targetNode.innerText == ' ' || targetNode.innerText ==
                '\n')
            {
                var oldRange = getSelection().getRangeAt(0);
                targetNode.innerText = "";

                // refocus the caret
                var selectionObject = getSelection();
                selectionObject.removeAllRanges();
                selectionObject.addRange(oldRange);
            }
        }

        function saveDescription()
        {
            var currentDescription = document.getElementById(
                "factDescription").innerText;
            var factNodes = document.getElementsByClassName("fact");
            for (var node = 0; node < factNodes.length; ++node)
            {
                if (factNodes[node].getAttribute("selected") == "true")
                {
                    factNodes[node].setAttribute("description",
                        currentDescription);
                }
            }
        }

        function updateSymbolMap(factConstructor)
        {
            Python.getSymbolList(factConstructor);
            var symbolList = Python.register;
            symbolList = symbolList.sort(function (leftArg, rightArg)
            {
                return parseInt(leftArg) - parseInt(rightArg);
            });
            Python.register = undefined;

            document.getElementById("symbolMap").innerHTML = "";
            var symbolNodes = document.getElementsByClassName("symbol");
            for (var symbol = 0; symbol < symbolList.length; ++symbol)
            {
                // check to see if the symbol has already been printed
                // in the dictionary
                var hasWritten = false;
                for (var symbolNode = 0; symbolNode <
                    symbolNodes.length; ++symbolNode)
                {
                    var symbolNumberString = symbolNodes[
```

```
                        symbolNode].children[0].innerText;
                    if (symbolNumberString == symbolList[
                        symbol].toString())
                    {
                        hasWritten = true;
                        var parent = document.createElement("div");
                        parent.className = "involvedSymbol";

                        var numberNode = document.createElement("div");
                        numberNode.className = "involvedSymbolNumber";
                        numberNode.innerText = "[" + symbolNumberString
                            + "]";
                        parent.appendChild(numberNode);

                        var definitionNode = document.createElement(
                            "div");
                        definitionNode.className =
                            "involvedSymbolDefinition placeholder";
                        definitionNode.setAttribute("placeholder",
                            "undefined")
                        definitionNode.innerText = symbolNodes[
                            symbolNode].children[1].innerText;
                        parent.appendChild(definitionNode);

                        document.getElementById(
                            "symbolMap").appendChild(parent);
                    }
                }

                // if the symbol hasn't been printed, print it
                if (!hasWritten)
                {
                    var parent = document.createElement("div");
                    parent.className = "involvedSymbol";

                    var numberNode = document.createElement("div");
                    numberNode.className = "involvedSymbolNumber";
                    numberNode.innerText = '[' + symbolList[symbol] +
                        ']';
                    parent.appendChild(numberNode);

                    var definitionNode = document.createElement("div");
                    definitionNode.className =
                        "involvedSymbolDefinition placeholder";
                    definitionNode.setAttribute("placeholder",
                        "undefined");
                    parent.appendChild(definitionNode);

                    document.getElementById("symbolMap").appendChild(
                        parent);
                }
            }
        }
```

```
var PLE = { facts: {}, symbols: {}, loader: {} };
PLE.facts.add = function ()
{
    var factNode = document.createElement("div");
    factNode.className = "fact placeholder";
    factNode.setAttribute("selected", "false");
    factNode.setAttribute("valid", "false");
    factNode.setAttribute("placeholder", "Enter a fact ...");
    factNode.setAttribute("description", "");
    document.getElementById("factList").appendChild(factNode);
    PLE.facts.toggleSelection(factNode);
    factNode.addEventListener("blur", function (event)
    {
        checkValidity(event.target);
        PLE.symbols.update();
    });
};
PLE.facts.clear = function ()
{
    if (confirm("Clear all facts?"))
    {
        var factNodes = document.getElementsByClassName("fact");
        if (factNodes.length > 0)
        {
            document.getElementById("factList").innerHTML = "";
            document.getElementById("symbolMap").innerHTML = "";
            document.getElementById("factDescription").innerText
                = "";
            document.getElementById("factDescription").
                setAttribute("editable", "false");
            document.getElementById("factDescription").
                setAttribute("placeholder", "");
            PLE.symbols.update();
        }
    }
};
PLE.facts.remove = function ()
{
    var factNodes = document.getElementsByClassName("fact");
    for (var fact = 0; fact < factNodes.length; ++fact)
    {
        if (factNodes[fact].getAttribute("selected") == "true")
        {
            factNodes[fact].parentNode.removeChild(factNodes[
                fact]);
            document.getElementById("symbolMap").innerHTML = "";
            document.getElementById(
                "factDescription").innerText = "";
            document.getElementById(
                "factDescription").setAttribute("editable",
                "false");
            document.getElementById(
```

```
                        "factDescription").setAttribute("placeholder",
                        "");
                }
            }
        };
        PLE.facts.toggleSelection = function (targetNode)
        {
            checkValidity(targetNode);
            if (targetNode.getAttribute("selected") == "false")
            {
                document.getElementById("factDescription").setAttribute(
                    "editable", "true");
                document.getElementById("factDescription").setAttribute(
                    "placeholder",
                    "Enter a description of this fact ... ");
                var oldSelectedNode = undefined;
                var newSelectedNode = undefined;

                // update fact highlighting
                var factList = document.getElementsByClassName("fact");
                for (var fact = 0; fact < factList.length; ++fact)
                {
                    if (factList[fact].getAttribute("selected") ==
                        "true")
                    {
                        oldSelectedNode = factList[fact];
                        factList[fact].setAttribute("selected",
                            "false");
                    }
                }
                newSelectedNode = targetNode;
                targetNode.setAttribute("selected", "true");

                // update description
                if (oldSelectedNode != undefined)
                {
                    var currentDescription = document.getElementById(
                        "factDescription").innerText;
                    oldSelectedNode.setAttribute("description",
                        currentDescription);
                }
                var newDescription = "";
                if (newSelectedNode != undefined)
                {
                    newDescription = newSelectedNode.getAttribute(
                        "description");
                }
                document.getElementById("factDescription").innerText =
                    newDescription;
            }

            // unhighlight current constituent
            var constituents = document.getElementsByClassName(
```

```
                "constituent");
            for (var constituent = 0; constituent < constituents.length;
                ++constituent)
            {
                if (constituents[constituent].getAttribute("selected")
                    == "true")
                {
                    constituents[constituent].setAttribute("selected",
                        "false");
                }
            }
            PLE.symbols.update();
        };
        PLE.symbols.add = function ()
        {
            var symbolNode = document.createElement("div");
            symbolNode.className = "symbol";
            symbolNode.setAttribute("selected", "false");
            symbolNode.setAttribute("eliminated", "false");

            var symbolNumberNode = document.createElement("div");
            symbolNumberNode.className =
                "symbolNumber editable symbolChild";
            symbolNumberNode.setAttribute("editable", "true");
            symbolNumberNode.innerText = "";
            symbolNode.appendChild(symbolNumberNode);

            var symbolNameNode = document.createElement("div");
            symbolNameNode.className =
                "symbolName symbolChild placeholder";
            symbolNameNode.setAttribute("placeholder",
                "Describe symbol here ...");
            symbolNode.appendChild(symbolNameNode);
            document.getElementById("symbolList").appendChild(
                symbolNode);
            PLE.symbols.toggleSelection(symbolNode);

            symbolNode.children[0].addEventListener("blur",
                PLE.symbols.update);
            symbolNode.children[1].addEventListener("blur",
                PLE.symbols.update);
        };
        PLE.symbols.clear = function ()
        {
            if (confirm("Clear all symbols?"))
            {
                var symbolNodes = document.getElementsByClassName(
                    "symbol");
                if (symbolNodes.length > 0)
                {
                    document.getElementById("symbolList").innerHTML =
                        "";
                    PLE.symbols.update();
```

```
                }
            }
        };
        PLE.symbols.remove = function ()
        {
            var symbolNodes = document.getElementsByClassName("symbol");
            if (symbolNodes.length > 0)
            {
                for (var symbol = 0; symbol < symbolNodes.length;
                    ++symbol)
                {
                    if (symbolNodes[symbol].getAttribute("selected") ==
                        "true")
                    {
                        symbolNodes[symbol].parentNode.removeChild(
                            symbolNodes[symbol]);
                    }
                }
            }
            PLE.symbols.update();
        };

        // Toggles whether or not the current symbol is quantum, and
        // should not appear in the final result therefore.
        PLE.symbols.toggleQuantum = function ()
        {
            var symbolNodes = document.getElementsByClassName("symbol");
            if (symbolNodes.length > 0)
            {
                for (var symbol = 0; symbol < symbolNodes.length;
                    ++symbol)
                {
                    if (symbolNodes[symbol].getAttribute("selected") ==
                        "true")
                    {
                        var oldAttribute = symbolNodes[
                            symbol].getAttribute("eliminated");
                        var newAttribute = "truefalse";
                        newAttribute = newAttribute.replace(
                            oldAttribute, "");
                        symbolNodes[symbol].setAttribute("eliminated",
                            newAttribute);
                    }
                }
            }
        }

        // Highlight targetNode, and deselect the currently selected
        // symbol.
        PLE.symbols.toggleSelection = function (targetNode)
        {
            while (targetNode.classList.contains("symbolChild"))
            {
```

```
                targetNode = targetNode.parentNode;
            }
            if (targetNode.getAttribute("selected") == "false")
            {
                var symbolList = document.getElementsByClassName(
                    "symbol");
                for (var symbol = 0; symbol < symbolList.length;
                    ++symbol)
                {
                    if (symbolList[symbol].getAttribute("selected") ==
                        "true")
                    {
                        symbolList[symbol].setAttribute("selected",
                            "false");
                    }
                }
                targetNode.setAttribute("selected", "true");
            }
            PLE.symbols.update();
        };

        // Display a list of involved symbols in the currently selected
        // fact or constituent.
        PLE.symbols.update = function ()
        {
            var currentSelectedFact = undefined;
            var factConstructor = undefined;

            // look for a selected fact
            var factList = document.getElementsByClassName("fact");
            for (var fact = 0; fact < factList.length; ++fact)
            {
                if (factList[fact].getAttribute("selected") == "true")
                {
                    currentSelectedFact = factList[fact];
                    factConstructor = currentSelectedFact.innerText;
                }
            }

            // look instead for a selected constituent if no fact is
            // found
            if (currentSelectedFact == undefined)
            {
                var constituentNodes = document.getElementsByClassName(
                    "constituent");
                for (var constituent = 0; constituent <
                    constituentNodes.length; ++constituent)
                {
                    if (constituentNodes[constituent].getAttribute(
                        "selected") == "true")
                    {
                        currentSelectedFact = constituentNodes[
                            constituent];
```

```
                        factConstructor = "0 = " +
                            currentSelectedFact.innerText;
                        factConstructor = factConstructor.replace("0/0",
                            "");
                        factConstructor = factConstructor.replace('+',
                            "");
                    }
                }
            }

            // update according to currentSelectedFact (or constituent)
            if (currentSelectedFact == undefined)
            {
                document.getElementById("symbolMap").innerHTML = "";
            }
            else
            {
                if (currentSelectedFact.getAttribute("valid") ==
                    "false")
                {
                    updateSymbolMap("");
                }
                else
                {
                    // the fact is at least pseudo-valid, and can have
                    // its symbol map checked.
                    updateSymbolMap(factConstructor);
                }
            }
        };
        PLE.loader.load = function (fileAddress)
        {
            Python.load(fileAddress);
            var wasSuccessful = (Python.register[0] == '1');
            var saveState = Python.register.slice(1);
            Python.register = undefined;

            if (wasSuccessful)
            {
                try
                {
                    var storedData = JSON.parse(saveState);
                    var relations = atob(storedData.relations);
                    document.getElementById("factList").innerHTML =
                        relations;
                    var symbols = atob(storedData.symbols);
                    document.getElementById("symbolList").innerHTML =
                        symbols;
                    alert("File opened successfully.");
                }
                catch (error)
                {
                    alert("The file is malformed.");
```

```
                }
            }
            else
            {
                alert("Could not find file.");
            }
        };
        PLE.loader.save = function (fileAddress)
        {
            saveDescription();
            deselectSelectedNodes();

            var relations = btoa(document.getElementById(
                "factList").innerHTML);
            var symbols = btoa(document.getElementById(
                "symbolList").innerHTML);
            var storedData = JSON.stringify({relations: relations,
                symbols: symbols
            });
            Python.save(storedData, fileAddress);
            var wasSuccessful = (Python.register[0] == '1');
            Python.register = undefined;

            if (wasSuccessful)
            {
                alert("File saved successfully.");
            }
            else
            {
                alert("The file path is invalid.");
            }
        };
    </script>
    <script name="back-end definitions" type="text/javascript">
        var Python = { cache: [], cacheInf: [], register: undefined };
        // cache to store massive results of processing, register for
        // less particular temp results, and cacheInf for a cache of
        // "partial" (implied) results

        Python.getSymbolList = function (factConstructor)
        {
            Bridge.getSymbolList(factConstructor);
        };
        Python.isValid = function (factConstructor)
        {
            Bridge.isValid(factConstructor)
        }
        Python.launchRequest = function ()
        {
            // append each fact
            var appendResults = [];
            var physicalFactList = document.getElementsByClassName(
                "fact");
```

```
            for (var fact = 0; fact < physicalFactList.length; ++fact)
            {
                var currentConstructor = physicalFactList[
                    fact].innerText;
                Bridge.addFact(currentConstructor);
                appendResults[appendResults.length] = this.cache;
            }
            var request = document.getElementById(
                "quaesitumInsertionPoint").innerText;

            // specify which symbols are in quantum states, and should
            // not be found in the final result
            eliminatedSymbols = getElimiatedSymbols();
            for (var symbol = 0; symbol < eliminatedSymbols.length;
                ++symbol)
            {
                Bridge.eliminateSymbol(eliminatedSymbols[symbol]);
            }
            Bridge.request(request);
        };
        Python.load = function (fileAddress)
        {
            Bridge.load(fileAddress);
        }

        // This function is volatile, and breaks with subtle
        // modifications of Python error messages.
        Python.mapError = function (errorMessage)
        {
            if (errorMessage ==
                "Failed to append fact: fact is malformed.")
            {
                // supress -- errors of this nature are made obvious to
                // the user through colouring
            }
            else if (errorMessage ==
                "Failed to set request: request is malformed.")
            {
                alert("The request is malformed.");
            }
            else if (errorMessage ==
                "Failed to declare environment: request is not defined."
                )
            {
                // suppress -- this is handled by the former case, this
                // error will not be found without a pairing with the
                // prior message
            }
            else
            {
                alert("error " + errorMessage);
            }
        };
```

```
        Python.onProcessorFinish = function ()
        {
            // write containers to #outBox
            document.getElementById("outBox").innerHTML = "";
            var titleNode = document.createElement("div");
            titleNode.className = "heading contentComponent upper";
            var requestText = document.getElementById(
                "quaesitumInsertionPoint").innerText;
            if (requestText == "")
            {
                requestText = 0;
            }
            titleNode.innerText = requestText + " = ";
            document.getElementById("outBox").appendChild(titleNode);

            var quantumNode = document.createElement("div");
            quantumNode.id = "quantumConstituentContainer";
            quantumNode.setAttribute("hiddenDiv", "true");
            document.getElementById("outBox").appendChild(quantumNode);

            var unityNode = document.createElement("div");
            unityNode.id = "determinateConstituentContainer";
            unityNode.setAttribute("hiddenDiv", "true");
            document.getElementById("outBox").appendChild(unityNode);

            document.getElementById("outBox").setAttribute("hiddenDiv",
                "false");

            // print the primary constituents, emptying the cache
            this.printConstituents();
            this.cache = [];

            // write container to #outBoxInf
            document.getElementById("outBoxInf").innerHTML = "";
            var titleNode = document.createElement("div");
            titleNode.className = "heading contentComponent upper";
            titleNode.innerText = "0 = ";
            document.getElementById("outBoxInf").appendChild(titleNode);

            var infiniteNode = document.createElement("div");
            infiniteNode.id = "infiniteConstituentContainer";
            document.getElementById("outBoxInf").appendChild(
                infiniteNode);
            if (this.cacheInf.length == 0)
            {
                document.getElementById("outBoxInf").setAttribute(
                    "hiddenDiv", "true");
            }
            else
            {
                document.getElementById("outBoxInf").setAttribute(
                    "hiddenDiv", "false");
            }
```

```
        // print the secondary constituents and empty the cache
        this.printInfiniteConstituents();
        this.cacheInf = [];

        // attach listeners to each of the constituents
        var nodesToListen = [document.getElementById(
            "quantumConstituentContainer"), document.getElementById(
            "determinateConstituentContainer"),
            document.getElementById("infiniteConstituentContainer")
            ];
        for (var node = 0; node < nodesToListen.length; ++node)
        {
            var nodeToListen = nodesToListen[node];
            if (nodeToListen != undefined)
            {
                nodeToListen.addEventListener("mousedown", function
                    (event)
                {
                    if (event.target.classList.contains(
                        "constituent"))
                    {
                        saveDescription();
                        deselectSelectedNodes();
                        event.target.setAttribute("selected",
                            "true");
                        PLE.symbols.update();
                    }
                });
            }
        }
    };

    // Print constituents that are either determinate or quantum,
    // having the respective coefficients of 1 or 0/0.
    Python.printConstituents = function ()
    {
        // iterate through the constituents, printing each
        var constituents = this.cache;
        var hasFoundDeterminate = false;
        for (var constituent = 0; constituent < constituents.length;
            ++constituent)
        {
            var currentConstituent = constituents[constituent];
            var isQuantum = undefined;
            var adornedConstituent = undefined;
            if (currentConstituent[0] == '1')
            {
                isQuantum = false;
                hasFoundDeterminate = true;
                if (document.getElementById(
                    "determinateConstituentContainer").getAttribute(
                    "hiddenDiv") == "true");
```

```
                    {
                        document.getElementById(
                            "determinateConstituentContainer"
                            ).setAttribute("hiddenDiv", "false");
                    }
                    adornedConstituent = currentConstituent.slice(1);
                }
                else // begins with 0/0
                {
                    isQuantum = true;
                    if (document.getElementById(
                        "quantumConstituentContainer").getAttribute(
                        "hiddenDiv") == "true");
                    {
                        document.getElementById(
                            "quantumConstituentContainer").setAttribute(
                            "hiddenDiv", "false");
                    }
                    adornedConstituent = currentConstituent;
                }
                if (constituent != constituents.length - 1)
                {
                    adornedConstituent += " + ";
                }
                else if (adornedConstituent.length == 0)
                {
                    adornedConstituent = "1";
                }
                var constituentNode = document.createElement("div");
                constituentNode.setAttribute("valid", "true");
                constituentNode.className = isQuantum ?
                    "quantumConstituent constituent" :
                    "determinateConstituent constituent";
                constituentNode.setAttribute("selected", "false");
                constituentNode.innerText = adornedConstituent;

                var destinationNode = isQuantum ?
                    document.getElementById(
                    "quantumConstituentContainer") :
                    document.getElementById(
                    "determinateConstituentContainer");
                destinationNode.appendChild(constituentNode);
            }

            // apply some conditional styles
            if (hasFoundDeterminate)
            {
                document.getElementById(
                    "determinateConstituentContainer").style.padding =
                    "0";
            }
            else
            {
```

```
                document.getElementById(
                    "quantumConstituentContainer").style.padding = "0";
                document.getElementById(
                    "quantumConstituentContainer").style.border =
                    "none";
            }
            if (this.cache.length == 0)
            {
                document.getElementById(
                    "determinateConstituentContainer").setAttribute(
                    "hiddenDiv", "false");

                var constituentNode = document.createElement("div");
                constituentNode.className =
                    "determinateConstituent constituent";
                constituentNode.setAttribute("selected", "false");
                document.getElementById(
                    "determinateConstituentContainer").style.paddingTop
                    = "0";
                constituentNode.innerText = "0";
                document.getElementById(
                    "determinateConstituentContainer").appendChild(
                    constituentNode);
            }
        }

        // Print constituents which have a coefficient of 1/0, termed
        // "implied relations".
        Python.printInfiniteConstituents = function ()
        {
            var constituents = this.cacheInf;
            for (var constituent = 0; constituent < constituents.length;
                ++constituent)
            {
                var currentConstituent = constituents[constituent];
                var adornedConstituent = currentConstituent.slice(3);
                // remove "1/0"

                if (constituent != constituents.length - 1)
                {
                    adornedConstituent += " + ";
                }
                else if (adornedConstituent.length == 0)
                {
                    adornedConstituent = "1";
                }

                var constituentNode = document.createElement("div");
                constituentNode.setAttribute("valid", "true");
                constituentNode.className =
                    "infiniteConstituent constituent";
                constituentNode.setAttribute("selected", "false");
                constituentNode.innerText = adornedConstituent;
```

```
                document.getElementById(
                    "infiniteConstituentContainer").appendChild(
                    constituentNode);
            }
        }
        Python.save = function (storedData, fileAddress)
        {
            Bridge.save(storedData, fileAddress);
        }
    </script>
    <script name="listeners" type="text/javascript">
        window.addEventListener("load", function ()
        {
            // standard listeners for left panel
            document.getElementById("factList").addEventListener(
                "mousedown", function (event)
            {
                if (event.target.classList.contains("fact"))
                {
                    PLE.facts.toggleSelection(event.target);
                }
            });
            document.getElementById("addFact").addEventListener("click",
                PLE.facts.add);
            document.getElementById("clearFacts").addEventListener(
                "click", PLE.facts.clear);
            document.getElementById("removeFact").addEventListener(
                "click", PLE.facts.remove);

            // standard listeners for right panel
            document.getElementById("symbolList").addEventListener(
                "mousedown", function (event)
            {
                if (event.target.classList.contains("symbol") ||
                    event.target.classList.contains("symbolChild"))
                {
                    PLE.symbols.toggleSelection(event.target);
                }
            });
            document.getElementById("addSymbol").addEventListener(
                "click", PLE.symbols.add);
            document.getElementById("clearSymbols").addEventListener(
                "click", PLE.symbols.clear);
            document.getElementById("removeSymbol").addEventListener(
                "click", PLE.symbols.remove);

            // non-standard listener for right panel
            document.getElementById("toggleQuantum").addEventListener(
                "click", PLE.symbols.toggleQuantum);

            // listener for all instigators of value change (live
            // validation)
```

```
            var EVENTS_CREATING_CHANGE = ["change", "cut", "drop",
                "keydown", "paste"];
            for (var eventNum = 0; eventNum <
                EVENTS_CREATING_CHANGE.length; ++eventNum)
            {
                var currentEvent = EVENTS_CREATING_CHANGE[eventNum];
                document.getElementById("symbolList").addEventListener(
                    currentEvent, function (event)
                {
                    setTimeout(function ()
                    {
                        formatSymbolNumber(event);
                    }, 0);
                });
                document.getElementById("factList").addEventListener(
                    currentEvent, function (event)
                {
                    if (event.target.classList.contains("fact"))
                    {
                        setTimeout(function ()
                        {
                            removeWhitespace(event.target);
                        }, 0);
                    }
                });
                document.getElementById(
                    "factDescription").addEventListener(currentEvent,
                    function (event)
                {
                    setTimeout(function ()
                    {
                        removeWhitespace(event.target);
                    }, 0);
                });
                document.getElementById("symbolList").addEventListener(
                    currentEvent, function (event)
                {
                    if (event.target.classList.contains("symbolName"))
                    {
                        setTimeout(function ()
                        {
                            removeWhitespace(event.target);
                        }, 0);
                    }
                });
                document.getElementById(
                    "quaesitumInsertionPoint").addEventListener(
                    currentEvent, function (event)
                {
                    setTimeout(function ()
                    {
                        removeWhitespace(event.target);
                    }, 0);
```

```
        });
    }

    // drop events must specifically be ignored due to their
    // shoddy standards and implementation
    window.addEventListener("dragover", function (event)
    {
        event.preventDefault();
    });

    // ignore any paste events into symbolNumber
    document.getElementById("symbolList").addEventListener(
        "paste", function (event)
    {
        if (event.target.classList != undefined)
        {
            if (event.target.classList.contains("symbolNumber"))
            {
                event.preventDefault();
            }
        }
    });

    // listener to prevent default context menu
    document.getElementsByTagName("body")[0].addEventListener(
        "contextmenu", function (event)
    {
        event.preventDefault();
    });

    // listener for form submission
    document.getElementById("request").addEventListener("click",
        Python.launchRequest);

    // listener for shortcut keys
    window.addEventListener("keypress", function (event)
    {
        if (event.ctrlKey)
        {
            if (event.keyCode == 19) // 's'
            {
                event.preventDefault();
                var fileAddress = prompt(
                    "Enter address to save to.");
                if (fileAddress != null)
                {
                    PLE.loader.save(fileAddress);
                }
            }
            else if (event.keyCode == 15) // 'o'
            {
                event.preventDefault();
                var fileAddress = prompt(
```

```
                            "Enter address to load from.");
                        if (fileAddress != null)
                        {
                            PLE.loader.load(fileAddress);
                        }
                    }
                }
            });
        });
    </script>
</head>
<body>
    <div id="tortilla">
        <div id="panelHolder">
            <div class="panelOuter">
                <div class="panelInner left">
                    <div class="content">
                        <div class="heading contentComponent upper">
                            Fact List
                        </div>
                        <div class="contentComponent middle">
                            <div id="factList" class="list"></div>
                        </div>
                        <div class="contentComponent lower">
                            <div class="buttonHolder">
                                <div id="clearFacts"
                                    class="button left">
                                    Clear All
                                </div>
                                <div id="removeFact"
                                    class="button middle">
                                    Remove
                                </div>
                                <div id="addFact" class="button right">
                                    Add
                                </div>
                            </div>
                            <!-- empty node such that .content's height
                                will be extended according to the
                                otherwise floated .buttons -->
                            <div style="color: rgba(0, 0, 0, 0.0);">
                                |
                            </div>
                        </div>
                    </div>
                </div>
                <div class="panelInner left offset">
                    <div class="content">
                        <div id="factDescription"
                             class="description editable placeholder">
                        </div>
                    </div>
                </div>
```

```
            </div>
            <div class="panelOuter">
                <div class="panelInner right">
                    <div class="content">
                        <div class="heading contentComponent upper">
                            Symbol Dictionary
                        </div>
                        <div class="contentComponent middle">
                            <div id="symbolList" class="list"></div>
                        </div>
                        <div class="contentComponent lower">
                            <div class="buttonHolder">
                                <div id="toggleQuantum"
                                     class="button left">
                                    Toggle Filter
                                </div>
                                <div id="clearSymbols"
                                     class="button middle">
                                    Clear All
                                </div>
                                <div id="removeSymbol"
                                     class="button middle">
                                    Remove
                                </div>
                                <div id="addSymbol"
                                     class="button right">
                                    Add
                                </div>
                            </div>
                            <!-- empty node such that .content's height
                                will be extended according to the
                                otherwise floated .buttons -->
                            <div style="color: rgba(0, 0, 0, 0.0);">
                                |
                            </div>
                        </div>
                    </div>
                </div>
                <div class="panelInner right offset">
                    <div class="content">
                        <div id="symbolMap" class="description"></div>
                    </div>
                </div>
            </div>
        </div>
        <div id="footHolder">
            <div class="footSpan">
                <div class="content">
                    <div id="request" class="button">
                        Make Request
                    </div>
                    <div id="quaesitumInsertionPoint"
                         class="editable placeholder"
```

```
                        placeholder="Insert quaesitum ..."
                        editable="true"></div>
                </div>
            </div>
            <div class="footSpan">
                <div id="outBox" class="content" hiddenDiv="true"></div>
                <div id="outBoxInf" class="content" hiddenDiv="true"
                    ></div>
            </div>
        </div>
    </div>
</body>
</html>
```

7.2. RUN_G_U_I.PY

```
# runGUI.py
# Nicholas Killeen,
# 31st July 2016.
# Runs a graphical user interface for the Propositional Logic Engine.

import os;

from PySide.QtCore import *;
from PySide.QtGui import *;
from PySide.QtWebKit import *;

import FactDatabase.FactDatabase as FactDatabase;
import Fraction.Fraction as Fraction;
import Interface.Interface as Interface;
import Processor.Processor as Processor;
import QuantumCounter.QuantumCounter as QuantumCounter;
import Relation.Relation as Relation
import runCLI;

MAX_RELATION_SYMBOLS_FOR_TESTING = 7;
# the maximum number of symbols that can be in a fact before the engine
# decides that the length of time involved in processing is detrimental
# to workflow
LARGEST_SYMBOL = 999;
# the largest valid symbol value

SAVE_PATH = "saves/";
FILE_TYPE = ".ple"

GInterface = None;
# global Interface object

def runGUI():
    # Class to bridge JavaScript and Python. Is a member of runGUI such
    # that the Object contents can be accessed without globals.
    class Bridge(QObject):
        @Slot(str)
```

```
        def addFact(this, relationConstructor):
            relationConstructor = runCLI._transpose(
                relationConstructor);
            factRelation = Relation.Relation(relationConstructor);
            GInterface.addFact(factRelation);

        @Slot(str)
        def eliminateSymbol(this, symbolNumber):
            GInterface.eliminateSymbol(int(symbolNumber));

        def evalJavaScript(this, script):
            contents.page().mainFrame().evaluateJavaScript(script);

        @Slot(str)
        def getSymbolList(this, relationConstructor):
            relationConstructor = runCLI._transpose(
                relationConstructor);
            isValidToCheckSymbols = _isRelationPseudoValid(
                relationConstructor);
            if (isValidToCheckSymbols):
                relation = Relation.Relation(relationConstructor);
                symbolList = relation.getSymbolList();
                this.evalJavaScript("Python.register = {0};".format(str(
                    symbolList)));
            else:
                # calling Relation.getSymbolList would throw an error
                this.evalJavaScript("Python.register = [];");

        # Method to be called manually after QObject.__init__.
        def initiate(this):
            global GInterface;
            GInterface = Interface.Interface(this.processError);

        @Slot(str)
        def isValid(this, relationConstructor):
            relationConstructor = runCLI._transpose(
                relationConstructor);
            relation = Relation.Relation(relationConstructor);
            isValidToCheckSymbols = _isRelationPseudoValid(
                relationConstructor);
            if (isValidToCheckSymbols):
                symbolList = relation.getSymbolList();
                if (len(symbolList) > 0 and max(symbolList) >
                    LARGEST_SYMBOL):
                    this.evalJavaScript("Python.register = false;");
                elif (len(symbolList) >
                    MAX_RELATION_SYMBOLS_FOR_TESTING):
                    this.evalJavaScript("Python.register = \"maybe\";");
                else:
                    isRelationValid = relation.isValid();
                    # this test completes the suite internal to
                    # Relation.isValid()
                    if (isRelationValid):
```

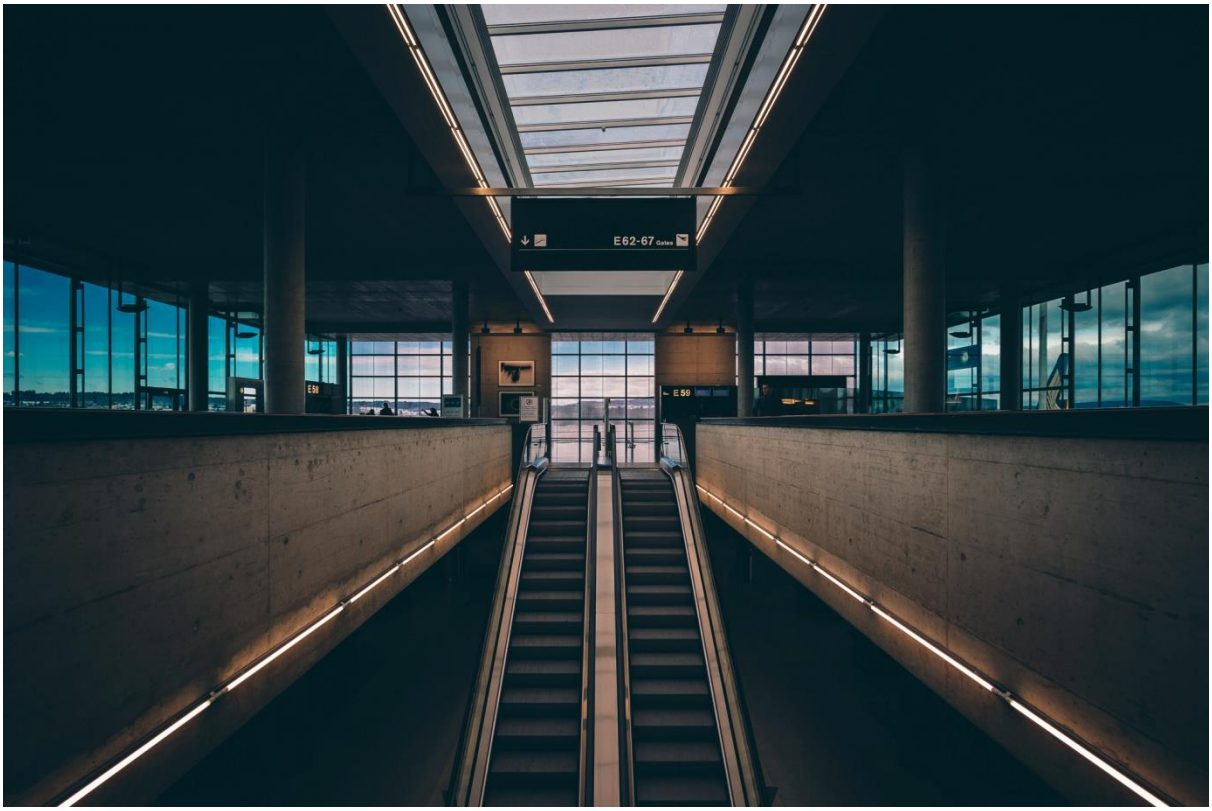
```
                        this.evalJavaScript("Python.register = true;");
                    else:
                        this.evalJavaScript("Python.register = false;");
            else:
                this.evalJavaScript("Python.register = false;");

        @Slot(str)
        def load(this, fileAddress):
            try:
                file = open("{0}{1}{2}".format(SAVE_PATH, fileAddress,
                    FILE_TYPE), 'r');
                fileString = file.read();

                while (fileString.count('\'') != 0):
                    fileString = fileString.replace('\'', "");
                    # the neglection of this replacement poses a serious
                    # security threat, whereby files can be manually
                    # engineered to behave otherwise like normal files,
                    # but can take partial or full control of the system
                    # through a process analagous to XSS attacks (this
                    # risk exists since eval(str) is being used to allow
                    # interaction between JS and Python)
                    # to handle this risk, any illegal characters are
                    # removed (as above), and processing then proceeds
                    # as normal

                this.evalJavaScript(
                    "Python.register = \"1\" + '{0}';".format(str(
                        fileString)));
                # '1' is a control character for whether or not the file
                # was successfully opened

                file.close();
                # the file doesn't need to be closed if it was never
                # instantiated correctly

            except Exception as error:
                this.evalJavaScript("Python.register = \"0\"");

        def processError(this, arg):
            this.evalJavaScript("Python.mapError(\"{0}\");".format(
                arg));

        @Slot(str)
        def request(this, requestStr):
            requestRelation = Relation.Relation(requestStr);
            if (requestRelation.isValid()):
                GInterface.setRequest(requestRelation);
                handler = GInterface.generateProcessor();
                handler();
                response = GInterface.retrieveResponse();
                constituentList = [];
                runCLI._formatSoughtInformation(response,
```

```
                    constituentList.append);

                # the constituentList array can be abbreviated at this
                # point by the general rule [0][1] + [0](1 - [1]) = [0],
                # but it is not a feature required for full
                # functionality

                # transfer constituents to JavaScript front-end
                constituentNumber = 0;
                numConstituents = len(constituentList);
                while (constituentNumber < numConstituents):
                    currentConstituent = constituentList[
                        constituentNumber];

                    isConstituentInfinite = False;
                    if (len(currentConstituent) > 1):
                        isConstituentInfinite = (
                            currentConstituent[0] == '1' and
                            currentConstituent[1] == '/');
                        # the truth of this implies that the constituent
                        # has a coefficient 1/0

                    if (isConstituentInfinite):
                        this.evalJavaScript("Python.cacheInf[Python.cac"
                            "heInf.length] = \"{0}\";".format(
                                currentConstituent));
                    else:
                        this.evalJavaScript("Python.cache[Python.cache."
                            "length] = \"{0}\";".format(
                                currentConstituent));
                    constituentNumber += 1;
                this.evalJavaScript("Python.onProcessorFinish();");

                # empty GInterface
                numFacts = len(GInterface.getFactList());
                while (numFacts != 0):
                    GInterface.deleteFact(0);
                    numFacts = len(GInterface.getFactList());
            else:
                this.processError("Failed to set request: request is ma"
                    "lformed.");

        @Slot(str, str)
        def save(this, data, fileAddress):
            _resolveSaveDirectory();
            try:
                file = open("{0}{1}{2}".format(SAVE_PATH, fileAddress,
                    FILE_TYPE), 'w');
                file.write(data);
                file.close();
            except Exception as error:
                this.evalJavaScript("Python.register = \"0\"");
```

```
    GUIhtml = open("PLE/GUI.html", 'r');
    htmlContent = GUIhtml.read();
    GUIhtml.close();
    htmlContent = htmlContent[htmlContent.find("<!DOCTYPE html>"):];
    # removes all header garbage before "<!DOCTYPE html>"

    # create a window
    PLE = QApplication([]);
    window = QWidget();
    window.setWindowTitle("Propositional Logic Engine");
    MARGIN_PX = -11;
    window.setContentsMargins(MARGIN_PX, MARGIN_PX, MARGIN_PX,
        MARGIN_PX);
    contents = QWebView();

    # bind the JavaScript and Python classes "Bridge"
    bridge = Bridge();
    bridge.initiate();
    contents.page().mainFrame().addToJavaScriptWindowObject("Bridge",
        bridge);

    # display the window
    contents.setHtml(htmlContent);
    layout = QVBoxLayout();
    layout.addWidget(contents);
    window.setLayout(layout);
    window.show();

    # run
    PLE.exec_();

# Determines whether a Relation is pseudo-valid, that is, does it meet
# all of the validity criteria that do not require lengthy evaluation.
def _isRelationPseudoValid(relationConstructor):
    isValid = Relation._areAllCharactersLegal(relationConstructor);
    if (isValid):
        isValid = Relation._areCombinationsLegal(relationConstructor);
    if (isValid):
        isValid = Relation._isNestingLegal(relationConstructor);
    if (isValid):
        isValid = Relation._areNumbersLegal(relationConstructor);
    return isValid;

# Instantiates the save/load file path if it does not already exist.
def _resolveSaveDirectory():
    if (not os.path.exists(SAVE_PATH)):
        os.makedirs(SAVE_PATH);
```

TESTING & EVALUATING

PART V.



1. DESKCHECKING

1.0. NOTES

Many of the modules written in the pseudocode so manifestly work, and it would therefore be entirely redundant to execute the planned test cases. Tests are only manually actuated when there is observed to be some benefit to the process, and thus, the reader may have some difficulty matching planned tests to executed tests -- but this is as warranted. The correctness of each module will be in some way justified, and options of this justification extend assertions of triviality, logical deductions, planned or additional test actuations, simulations, and a plethora of other methods.

1.1. VOID FACT_DATABASE::APPEND_FACT(RELATION FACT)

The algorithm written here so clearly works -- all logic has been delegated to an already present append method. There is no need for there to be any formal proof: the appendFact functions process is in literal form "append a fact to the list of facts", and the argument of correctness is furnished by any sane reasoning.

1.2. VOID FACT_DATABASE::DELETE_FACT(INT FACT_NUMBER)

The only dynamic component of this module is one that can easily be desk checked so as to ensure correctness:

```
00 FOR fact = 0 TO numFacts - 1
01     IF (fact < factNumber) THEN
02         sourceIndex = fact
03     ELSE
04         sourceIndex = fact + 1
05     ENDIF
06     this.appendFact(oldFactList[sourceIndex])
07 NEXT fact
08
```

FactDatabase::deleteFact(0) on [fact1]						
Line	factNumber	oldFactList	numFacts	fact	sourceIndex	this.factList
00	0	[fact1]	1	0	--	[]
08	0	[fact1]	1	0	--	[]

Expected	Actual	Result
this.factList = []	this.factList = []	Success

FactDatabase::deleteFact(0) on [fact1, fact2]						
Line	factNumber	oldFactList	numFacts	fact	sourceIndex	this.factList
00	0	[fact1, fact2]	2	0	--	[]
01	0	[fact1, fact2]	2	0	--	[]
04	0	[fact1, fact2]	2	0	1	[]
06	0	[fact1, fact2]	2	0	1	[fact2]
07	0	[fact1, fact2]	2	1	1	[fact2]

00	0	[fact1, fact2]	2	1	1	[fact2]
08	0	[fact1, fact2]	2	1	1	[fact2]
Expected			Actual		Result	
this.factList = [fact2]			this.factList = [fact2]		Success	

FactDatabase::deleteFact(1) on [fact1, fact2]						
Line	factNumber	oldFactList	numFacts	fact	sourceIndex	this.factList
00	1	[fact1, fact2]	2	0	--	[]
01	1	[fact1, fact2]	2	0	--	[]
02	1	[fact1, fact2]	2	0	0	[]
06	1	[fact1, fact2]	2	0	0	[fact1]
07	1	[fact1, fact2]	2	1	0	[fact1]
00	1	[fact1, fact2]	2	1	0	[fact1]
08	1	[fact1, fact2]	2	1	0	[fact1]
Expected			Actual		Result	
this.factList = [fact1]			this.factList = [fact1]		Success	

A more complex example should also be tested, despite being unlisted in the planned test cases:

FactDatabase::deleteFact(1) on [f1, f2, f3]						
Line	factNumber	oldFactList	numFacts	fact	sourceIndex	this.factList
00	1	[f1, f2, f3]	3	0	--	[]
01	1	[f1, f2, f3]	3	0	--	[]
02	1	[f1, f2, f3]	3	0	0	[]
06	1	[f1, f2, f3]	3	0	0	[f1]
07	1	[f1, f2, f3]	3	1	0	[f1]
00	1	[f1, f2, f3]	3	1	0	[f1]
01	1	[f1, f2, f3]	3	1	0	[f1]
04	1	[f1, f2, f3]	3	1	2	[f1]
06	1	[f1, f2, f3]	3	1	2	[f1, f3]
07	1	[f1, f2, f3]	3	2	2	[f1, f3]
00	1	[f1, f2, f3]	3	2	2	[f1, f3]
08	1	[f1, f2, f3]	3	2	2	[f1, f3]
Expected			Actual		Result	
this.factList = [f1, f3]			this.factList = [f1, f3]		Success	

The only other aspect of this subroutine is the input validation, and simply by observing the following line it can be seen as being correct:

```
IF factNumber < 0 OR factNumber >= numFacts THEN
```

1.3. FACT_DATABASE::FACT_DATABASE(VOID)

No tests were written for this module, and this is in line with what ought to be. The constructor, here, simply allocates memory for an empty factList, and there is no conceivable way that it can be incorrect.

1.4. RELATION FACT_DATABASE::GET_FACT_AGGREGATION(VOID)

The dynamic portion of the subroutine can be labelled as follows, and the planned data tested:

```
00 FOR fact = 0 TO numFacts
01     productOfPriorFacts = new Relation("1")
02     FOR encounteredFact = 0 to fact
03         productOfPriorFacts *= UNITY - this.factList[encounteredFact]
04     NEXT encounteredFact
05     currentFact = this.factList[fact]
06     aggregate += currentFact * productOfPriorFacts
07 NEXT fact
08
```

FactDatabase::getFactAggregation() on []							
Line	this.factList	numFacts	fact	enc.Fact	ΠPriorFacts	currentFact	aggregate
00	[]	0	0	--	--	--	0
08	[]	0	0	--	--	--	0

Expected	Actual	Result
aggregate = 0	aggregate = 0	Success

FactDatabase::getFactAggregation() on [fact1]							
Line	this.factList	numFacts	fact	enc.Fact	ΠPriorFacts	currentFact	aggregate
00	[fact1]	1	0	--	--	--	0
01	[fact1]	1	0	--	1	--	0
02	[fact1]	1	0	0	1	--	0
05	[fact1]	1	0	0	1	fact1	0
06	[fact1]	1	0	0	1	fact1	fact1
07	[fact1]	1	1	0	1	fact1	fact1
00	[fact1]	1	0	0	1	fact1	fact1
08	[fact1]	1	0	0	1	fact1	fact1

Expected	Actual	Result
aggregate = fact1	aggregate = fact1	Success

FactDatabase::getFactAggregation() on [f1, f2]							
Line	this.factList	numFacts	fact	enc.Fact	ΠPriorFacts	currentFact	aggregate
00	[f1, f2]	2	0	--	--	--	0
01	[f1, f2]	2	0	--	1	--	0
02	[f1, f2]	2	0	0	1	--	0
05	[f1, f2]	2	0	0	1	f1	0
06	[f1, f2]	2	0	0	1	f1	f1
07	[f1, f2]	2	1	0	1	f1	f1
00	[f1, f2]	2	1	0	1	f1	f1
01	[f1, f2]	2	1	0	1	f1	f1
02	[f1, f2]	2	1	0	1	f1	f1
03	[f1, f2]	2	1	0	(1 - f1)	f1	f1
04	[f1, f2]	2	1	1	(1 - f1)	f1	f1
02	[f1, f2]	2	1	1	(1 - f1)	f1	f1
05	[f1, f2]	2	1	1	(1 - f1)	f2	f1

06	[f1, f2]	2	1	1	(1 - f1)	f2	f1 + f2(1 - f1)
07	[f1, f2]	2	2	1	(1 - f1)	f2	f1 + f2(1 - f1)
00	[f1, f2]	2	2	1	(1 - f1)	f2	f1 + f2(1 - f1)
08	[f1, f2]	2	2	1	(1 - f1)	f2	f1 + f2(1 - f1)
Expected			Actual			Result	
aggregate = f1 + f2(1 - f1)			aggregate = f1 + f2(1 - f1)			Success	

FactDatabase::getFactAggregation() on [f1, f2, f3]							
Line	this.factList	numFacts	fact	enc.Fact	ΠPriorFacts	currentFact	aggregate
00	[f1, f2, f3]	3	0	--	--	--	0
01	[f1, f2, f3]	3	0	--	1	--	0
02	[f1, f2, f3]	3	0	0	1	--	0
05	[f1, f2, f3]	3	0	0	1	f1	0
06	[f1, f2, f3]	3	0	0	1	f1	f1
07	[f1, f2, f3]	3	1	0	1	f1	f1
00	[f1, f2, f3]	3	1	0	1	f1	f1
01	[f1, f2, f3]	3	1	0	1	f1	f1
02	[f1, f2, f3]	3	1	0	1	f1	f1
03	[f1, f2, f3]	3	1	0	(1 - f1)	f1	f1
04	[f1, f2, f3]	3	1	1	(1 - f1)	f1	f1
02	[f1, f2, f3]	3	1	1	(1 - f1)	f1	f1
05	[f1, f2, f3]	3	1	1	(1 - f1)	f2	f1
06	[f1, f2, f3]	3	1	1	(1 - f1)	f2	f1 + f2(1 - f1)
07	[f1, f2, f3]	3	2	1	(1 - f1)	f2	f1 + f2(1 - f1)
00	[f1, f2, f3]	3	2	1	(1 - f1)	f2	f1 + f2(1 - f1)
01	[f1, f2, f3]	3	2	1	1	f2	f1 + f2(1 - f1)
02	[f1, f2, f3]	3	2	0	1	f2	f1 + f2(1 - f1)
03	[f1, f2, f3]	3	2	0	(1 - f1)	f2	f1 + f2(1 - f1)
04	[f1, f2, f3]	3	2	1	(1 - f1)	f2	f1 + f2(1 - f1)
02	[f1, f2, f3]	3	2	1	(1 - f1)	f2	f1 + f2(1 - f1)
03	[f1, f2, f3]	3	2	1	(1 - f1)(1 - f2)	f2	f1 + f2(1 - f1)
04	[f1, f2, f3]	3	2	2	(1 - f1)(1 - f2)	f2	f1 + f2(1 - f1)
02	[f1, f2, f3]	3	2	2	(1 - f1)(1 - f2)	f2	f1 + f2(1 - f1)
05	[f1, f2, f3]	3	2	2	(1 - f1)(1 - f2)	f3	f1 + f2(1 - f1)
06	[f1, f2, f3]	3	2	2	(1 - f1)(1 - f2)	f3	f1 + f2(1 - f1) + f3(1 - f1)(1 - f2)
07	[f1, f2, f3]	3	3	2	(1 - f1)(1 - f2)	f3	f1 + f2(1 - f1) + f3(1 - f1)(1 - f2)
00	[f1, f2, f3]	3	3	2	(1 - f1)(1 - f2)	f3	f1 + f2(1 - f1) + f3(1 - f1)(1 - f2)
08	[f1, f2, f3]	3	3	2	(1 - f1)(1 - f2)	f3	f1 + f2(1 - f1) + f3(1 - f1)(1 - f2)
Expected			Actual			Result	
aggregate = f1 + f2(1 - f1) + f3(1 - f2)(1 - f1)			aggregate = f1 + f2(1 - f1) + f3(1 - f2)(1 - f1)			Success	

In the cases above, "phantom memory" may be noticed, for example, the currentFact variable persists despite going out of scope. This is for simplicity of testing -- it ought to have no effect on the results.

1.5. RELATION[] FACT_DATABASE::LIST_FACTS(VOID),
1.6. FRACTION::FRACTION(RELATION NUMERATOR, RELATION DENOMINATOR),
1.7. RELATION FRACTION::GET_DENOMINATOR(VOID),
1.8. RELATION FRACTION::GET_NUMERATOR(VOID)

Each of these subroutines are extraordinarily simple interface functions to get or set local variables. The FactDatabase::ListFacts function simply returns a locally stored array, and the whole Fraction class only serves the purpose of tuple storage. There is, in fact, a way to render the inclusion of a Fraction class frivolous, by the marvellous tools available in a Boolean environment:

δ_1 and δ_2 must be separately encoded, thus, they can be retrieved by using different flags: θ_1 and θ_2, $\therefore$ the encoded information can be represented by $\delta_3(\theta_1, \theta_2)$ -- $\delta_3(\theta_1, \theta_2) = \theta_1\delta_1 + \theta_2\delta_2$, but, developing with respect to θ_1 and θ_2 reveals an area where optimisation is applicable:

$$\delta_3(\theta_1, \theta_2) = \theta_1\theta_2(\delta_1 + \delta_2) + \theta_1(1 - \theta_2)\delta_1 + (1 - \theta_1)\theta_2\delta_2 + (1 - \theta_1)(1 - \theta_2)(0)$$

It can easily be noticed that exactly half of the coefficients are wasted: the first $(\delta_1 + \delta_2)$ is nonsensical, and the final coefficient 0 vanishes. Since half of the coefficients are wasted, one of the parameters can be removed:

$$\delta_3(\theta) = \theta\delta_1 + (1 - \theta)\delta_2$$

And this is the form of the equation in its most simple developed form, thus, a fraction can be encoded in a unary function of type delta, with the parameter being representative of whether the numerator or denominator is being requested.

But such a reduction adds unnecessary complexity. Despite this, its demonstration serves to put into perspective the simplicity of the Fraction class.

1.9. RELATION IO_STREAM::GET_FACT(VOID)

This function is mostly concerned with string manipulation, which, while still able to be deskchecked, is more easily testable through simulation. Were this function to be deskchecked, a string would have to be scanned linearly -- one character at a time -- which is quite time consuming. JavaScript has been chosen for the simulation as by its simplicity in terms of string manipulation, and a sample implementation follows:

```
function getFact(factConstructor)
{
    var stringLength = factConstructor.length;
    var LHS = "";
    var RHS = "";
    var hasEncounteredCopula = false;
    for (var character = 0; character < stringLength; ++character)
    {
        var currentCharacter = factConstructor[character];
        if (currentCharacter == '=')
```

```
        {
            hasEncounteredCopula = true;
        }
        else if (currentCharacter != ' ')
        {
            if (hasEncounteredCopula)
            {
                RHS += currentCharacter;
            }
            else
            {
                LHS += currentCharacter;
            }
        }
    }
    var fact = ""
    var subject = LHS
    var predicate = RHS
    fact += subject + '*' + "(1-" + predicate + ')';
    fact += '+' + predicate + '*' + "(1-" + subject + ')';
    return fact;
}
```

Calling

```
getFact("guacamole");
```

Returns

```
"guacamole*(1-)+*(1-guacamole)"
```

Which almost looks like it makes sense, but there is definitely no expectation of success seeing as the input validation process has not been implemented.

To test the additional functionality of whitespace removal:

```
getFact("guacamole") == getFact("   g  u a camol e   ");
```

Testing now the abstract case that was predicted to succeed (also implementing whitespace):

```
getFact(" <s ubje  ct> = <pr edic at  e>")
```

Returns

```
"<subject>*(1-<predicate>)+<predicate>*(1-<subject>)"
```

Which is in agreement with the expected result.

1.10. INT[] IO_STREAM::GET_FILTER(VOID)

The dynamic part of this routine follows -- and although it is fairly obviously correct, its testing is not majorly out of the way --:

```
00 FOR symbol = 0 TO numSymbols
01     IF isHighlighted[symbol] THEN
02         Append symbol to highlightedSymbols
03     ENDIF
04 NEXT symbol
05
```

IOStream::getFilter() on [false, true, false, true]				
Line	isHighlighted	numSymbols	symbol	highlightedSymbols
00	[false, true, false, true]	4	0	[]
01	[false, true, false, true]	4	0	[]
04	[false, true, false, true]	4	1	[]
00	[false, true, false, true]	4	1	[]
01	[false, true, false, true]	4	1	[]
02	[false, true, false, true]	4	1	[1]
04	[false, true, false, true]	4	2	[1]
00	[false, true, false, true]	4	2	[1]
01	[false, true, false, true]	4	2	[1]
04	[false, true, false, true]	4	3	[1]
00	[false, true, false, true]	4	3	[1]
01	[false, true, false, true]	4	3	[1]
02	[false, true, false, true]	4	3	[1, 3]
04	[false, true, false, true]	4	4	[1, 3]
00	[false, true, false, true]	4	4	[1, 3]
05	[false, true, false, true]	4	4	[1, 3]

Expected	Actual	Result
highlightedSymbols = [1, 3]	highlightedSymbols = [1, 3]	Success

This result is quite obviously generalisable.

1.11. RELATION IO_STREAM::GET_REQUEST(VOID)

This module wholly reuses the code in `1.9., and the fact that they do not call a common function (as suggested in the correlating section from `III.) is a blatant design error.

1.12. VOID IO_STREAM::GIVE_SOUGHT_INFORMATION(FRACTION SOUGHT_INFORMATION)

Due to the complexity of this function (which is a design fallacy), tests are quite expensive. According to the operational analysis (`III. 5.1.), the complexity is at $O(n \cdot 2^n)$. This would be an optimal place for simulation, but, this function has several dependencies which would also have to be implemented or spoofed for a simulation. To manually run through the nine planned test cases would take a great extent of time, and thus, there is no easy way to be absolutely certain that the logic of this routine is correct.

The potential for fallibility, although objectively bad, isn't all that detrimental. Under these headings, tests are being carried out on draft components, and these routines will not find their way into the system without a great deal of modification, and certainly not without dedicated unit testing (`III. 3.4.); even though this algorithm clearly fails to meet the quality descriptor of testability, the quality of the solution ought not to be directly diminished with appropriate precautions. What can be tested without much difficulty is one of the called modules getCombinedSymbolList (there ought to be many more delegate calls such as this in optimal design). Most of this routine is dynamic, so the whole algorithm will be cited and checked:

```
00 BEGIN getCombinedSymbolList(relationA, relationB)
01     involvedSymbols = relationA.getInvolvedSymbolList()
02     Set numSymbolsInA to number of elements in involvedSymbols
03     symbolsInB = relationB.getInvolvedSymbolList()
04     Set numSymbolsInB to number of elements in symbolsInB
05     FOR symbolInB = 0 TO numSymbolsInB
06         currentSymbol = symbolsInB[symbolInB]
07         collisionEncountered = false
08         FOR symbolInA = 0 TO numSymbolsInA
09             IF currentSymbol == involvedSymbols[symbolInA] THEN
10                 collisionEncountered = true
11             ENDIF
12         NEXT symbolInA
13         IF NOT collisionEncountered THEN
14             Append currentSymbol to involvedSymbols
15         ENDIF
16     NEXT symbolInB
17     RETURN involvedSymbols
18 END getCombinedSymbolList
```

getCombinedSymbolList("[0][2][4]", "[0][3][6]")								
Line	numSym.A	numSym.B	sym.sInB	sym.InB	currentSym.	sym.InA	collisionEncountered	involvedSym.
00	--	--	--	--	--	--	--	--
01	--	--	--	--	--	--	--	[0, 2, 4]
02	3	--	--	--	--	--	--	[0, 2, 4]
03	3	--	[0, 3, 6]	--	--	--	--	[0, 2, 4]
04	3	3	[0, 3, 6]	--	--	--	--	[0, 2, 4]
05	3	3	[0, 3, 6]	0	--	--	--	[0, 2, 4]
06	3	3	[0, 3, 6]	0	0	--	--	[0, 2, 4]
07	3	3	[0, 3, 6]	0	0	--	false	[0, 2, 4]
…								
13	3	3	[0, 3, 6]	0	0	3	true	[0, 2, 4]
16	3	3	[0, 3, 6]	1	0	3	true	[0, 2, 4]
05	3	3	[0, 3, 6]	1	0	3	true	[0, 2, 4]
06	3	3	[0, 3, 6]	1	3	3	true	[0, 2, 4]
07	3	3	[0, 3, 6]	1	3	3	false	[0, 2, 4]
…								
13	3	3	[0, 3, 6]	1	3	3	false	[0, 2, 4]

14	3	3	[0, 3, 6]	1	3	3	false	[0, 2, 4, 3]
16	3	3	[0, 3, 6]	2	3	3	false	[0, 2, 4, 3]
05	3	3	[0, 3, 6]	2	3	3	false	[0, 2, 4, 3]
06	3	3	[0, 3, 6]	2	6	3	false	[0, 2, 4, 3]
07	3	3	[0, 3, 6]	2	6	3	false	[0, 2, 4, 3]
...								
13	3	3	[0, 3, 6]	2	6	3	false	[0, 2, 4, 3]
14	3	3	[0, 3, 6]	2	6	3	false	[0, 2, 4, 3, 6]
16	3	3	[0, 3, 6]	3	6	3	false	[0, 2, 4, 3, 6]
05	3	3	[0, 3, 6]	3	6	3	false	[0, 2, 4, 3, 6]
17	3	3	[0, 3, 6]	3	6	3	false	[0, 2, 4, 3, 6]

Expected	Actual	Result
involvedSymbolList = [0, 2, 3, 4, 6]	involvedSymbolList = [0, 2, 4, 3, 6]	Success (but notice the entropy)

Three omissions have been made, greatly reducing the size of the tape. The omitted logic (lines 8 to 12) quite obviously is functional, and needn't have its accuracy furnished by a walkthrough.

The routine wholly -- giveSoughtInformation -- is still far from being actually untestable. It is still beneficial to at least examine the algorithm (although tedious) in order to get some idea as to whether it is or is not correct.

The variant portion of the routine giveSoughtInformation follows, accompanied then by an exploration of possible execution paths:

```
00 FOR constituent = 0 TO numConstituents
01     Initialise this.factList to an empty array
02     Copy numerator into numeratorCopy
03     Copy denominator into denominatorCopy
04     numeratorCopy.insertArguments(coefficientMap.read())
05     numeratorValue = numeratorCopy.evaluate()
06     denominatorCopy.insertArguments(coefficientMap.read())
07     denominatorValue = denominatorCopy.evaluate()
08
09     IF numeratorValue == denominatorValue
10         displayableConstituent = ""
11         FOR symbol = 0 TO numSymbols
12             IF coefficientMap.read()[symbol] == 0 THEN
13                 displayableConstituent += "[", symbol, "]"
14             ELSEIF coefficientMap.read()[symbol] == 1 THEN
15                 displayableConstituent += "(1 - [", symbol, "])"
16             ENDIF
17         NEXT symbol
18         IF numeratorValue == 1 THEN
19             IF displayableConstituent == "" THEN
20                 Display "1"
21             ELSE
22                 Display displayableConstituent
23             ENDIF
24         ELSE
```

```
25              Display "0/0", displayableConstituent
26          ENDIF
27      ENDIF
28      coefficientMap.increment()
29 NEXT constituent
```

There are four possible outcomes per iteration of the whole loop: the coefficient is of type $\frac{theta}{theta}$, and simple deduction allows the inference to be made that there are four possible outcomes, and thus, four possible ways in which the loop can execute. The logic of the algorithm can therefore be modularised

- Condition One. The coefficient takes the form 1/1. The constituent ought to be printed to the screen with the coefficient of unity, or, if the constituent is empty, the sole value of unity ought to be printed.

- Condition Two. The coefficient takes the form 1/0. The constituent ought to be ignore, and nothing printed.

- Condition Three. The coefficient takes the form 0/1. The constituent ought to be ignored, and nothing printed.

- Condition Four. The coefficient takes the form of 0/0. The constituent ought to be printed to the screen with an indeterminate coefficient, or, if the constituent contains no value, 0/0 ought to be printed as is.

Conditions Two and Three can quickly be seen to work (the examination of One is postponed), for they do not satisfy the least derived condition on line 09, and the program moves on to the next constituent and rectifies the coefficientMap.

Under condition Four, displayableConstituent is generated (the correctness of which will discussed later), the condition on line 18 is evaluated to false, and thus, execution skips to line 25, where the sought response is given.

Under condition One, there is a division of two outcomes. Consistently, displayableConstituent is generated, and the conditional at 18 resolves to true, thus, the division follows:

```
19 IF displayableConstituent == "" THEN
20     Display "1"
21 ELSE
22     Display displayableConstituent
23 ENDIF
```

It is not difficult to see that this is the hoped for outcome, and thus, the correctness of the majority of the algorithm falls only to the generation of the string displayableConstituent, which can be manually gazed upon quite painlessly:

```
11 FOR symbol = 0 TO numSymbols
12     IF coefficientMap.read()[symbol] == 0 THEN
13         displayableConstituent += "[", symbol, "]"
```

```
14     ELSEIF coefficientMap.read()[symbol] == 1 THEN
15         displayableConstituent += "(1 - [", symbol, "])"
16     ENDIF
17 NEXT symbol
```

Line 11 is manifestly correct: the length of each constituent is given by |φ(r)|, namely, the number of symbols in the relation that is being developed. The two conditionals are also clearly infallible -- for every symbol, if the respective coefficient n is in it for 0, the constituent should have as a member (1 - [n]), and if the coefficient of n is in it for 1, the constituent ought to have [n] as being fully factorisable, according to the simple form:

$$\delta(\theta) = (1 - \theta)\delta(0) + \theta\delta(1)$$

The logic of the module is whence proven as quite likely correct, certainly to the degree that is necessary.

1.13. IO_STREAM::IO_STREAM(VOID)

This algorithm is intended to do nothing. The algorithm does nothing. Q.E.D.

1.14. RELATION MAIN::ADD_FACT(FACT_DATABASE DATABASE, RELATION FACT)

This method is merely a strange migration of the FactDatabase::appendFact method, which really just defers to an append function which already exists in most source languages, or else, easily implementable.

1.15. RELATION MAIN::APPLY_FILTER(RELATION FACT_AGGREGATION, INT[] SYMBOLS_TO_ELIMINATE)

The simplicity of the module is conducive to easy completion of the planned tests:

```
00 FOR symbolHash = 0 TO numSymbolsToEliminate
01     symbolID = symbolsToEliminate[symbolHash]
02     newFactAggregation.eliminateSymbol(symbolID)
03 NEXT symbolHash
04
```

Main::applyFilter("(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]", [])					
Line	symbolsToElim.	numSymbolsToElim.	symbolHash	symbolID	newFactAggregation
00	[]	0	0	--	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"
04	[]	0	0	--	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"

Expected	Actual	Result
newFactAggregation = "(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	newFactAggregation = "(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	Success

Main::applyFilter("(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]", [4])					
Line	symbolsToElim.	numSymbolsToElim.	symbolHash	symbolID	newFactAggregation
00	[4]	1	0	--	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"
01	[4]	1	0	4	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"
02	[4]	1	0	4	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"
03	[4]	1	1	4	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"
00	[4]	1	1	4	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"
04	[4]	1	1	4	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"

Expected	Actual	Result
newFactAggregation = "(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	newFactAggregation = "(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	Success

Main::applyFilter("(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]", [3])					
Line	symbolsToElim.	numSymbolsToElim.	symbolHash	symbolID	newFactAggregation
00	[3]	1	0	--	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"
01	[3]	1	0	3	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"
02	[3]	1	0	3	"[0][1]+[2](1-[0][1])"
03	[3]	1	1	3	"[0][1]+[2](1-[0][1])"
00	[3]	1	1	3	"[0][1]+[2](1-[0][1])"
04	[3]	1	1	3	"[0][1]+[2](1-[0][1])"

Expected	Actual	Result
newFactAggregation = "[0][1]+[2](1-[0][1])"	newFactAggregation = "[0][1]+[2](1-[0][1])"	Success

Main::applyFilter("(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]", [2, 3])					
Line	symbolsToElim.	numSymbolsToElim.	symbolHash	symbolID	newFactAggregation
00	[2, 3]	2	0	--	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"
01	[2, 3]	2	0	2	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"
02	[2, 3]	2	0	2	"[0][1]+[2](1-[0][1])"
03	[2, 3]	2	1	2	"[0][1]+[2](1-[0][1])"
00	[2, 3]	2	1	2	"[0][1]+[2](1-[0][1])"
01	[2, 3]	2	1	3	"[0][1]+[2](1-[0][1])"
02	[2, 3]	2	1	2	"[0][1]"
03	[2, 3]	2	2	2	"[0][1]"
00	[2, 3]	2	2	2	"[0][1]"
04	[2, 3]	2	2	2	"[0][1]"

Expected	Actual	Result
newFactAggregation = "[0][1]"	newFactAggregation = "[0][1]"	Success

All of the planned tests needn't be executed because of the degree of simplicity of the algorithm. The results can be quite successfully extrapolated to absolute correctness, but this function, of course, has as a dependant the Relation::eliminateSymbol routine, wherein the accuracy of this module is wholly based, for this function recursively calls eliminateSymbol as a delegate.

1.16. MAIN(VOID)

To test the main method would be to test the system as a whole, and hereto, the system has not been assembled wholly, merely, individual components generated. The testing of the solution wholly will only take place once an implementation has been created.

1.17. FRACTION MAIN::REQUEST_INFORMATION(RELATION SUBJECT, FACT_DATABASE DATABASE)

This module shares most of its logic with IOStream::giveSoughtInformation, and hence, there exists no necessity of testing. This algorithm serves mainly to tie other smaller routines together.

1.18. VOID RELATION::ELIMINATE_SYMBOL(INT SYMBOL_HASH)

```
00 argumentMap[symbolHash] = 0
01 relationWith0.insertArguments(argumentMap)
02 argumentMap[symbolHash] = 1
03 relationWith1.insertArguments(argumentMap)
04 productOfElimination = relationWith0 * relationWith1
05
```

Relation::eliminateSymbol(4) on "(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"					
Line	argumentMap	symbolHash	relationWith0	relationWith1	ΠOfElimination
00	[%, %, %, %, 0]	4	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	--
01	[%, %, %, %, 0]	4	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	--
02	[%, %, %, %, 1]	4	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	--
03	[%, %, %, %, 1]	4	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	--
04	[%, %, %, %, 1]	4	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"
05	[%, %, %, %, 1]	4	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	"(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"

Expected	Actual	Result
productOfElimination = "(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	productOfElimination = "(1-[3])[0][1]+(1-[3])[2](1-[0][1])+[3]"	Success

There can instantly be noticed potential for error, depending on how arrays are implemented in the source language. Perhaps with an argumentMap of [%], and a symbolHash of 2, argumentMap will be set to contain [%, GARBAGE, 0] -- and there is therefore a memory leak. The solution to this lies in the interpretability of the overhead line:

```
Set largestSymbol to the greatest value in involvedSymbolList
```

If, then, the largest value of either largestSymbol or symbolHash is taken and reallocated to largestSymbol, any potential garbage reading will be avoided. One more of the planned test cases is enough to furnish the functionality of this routine:

Relation::eliminateSymbol(10) on "[10][2][3]+[10](1-[2])(1-[3])+(1-[10])(1-[2])"					
Line	argumentMap	sym.Hash	relationWith0	relationWith1	ΠOfElimination
00	[%, %, %, %, %, %,%,%,%, %, 0]	10	"[10][2][3]+[10](1-[2])(1-[3])+(1-[10])(1-[2])"	"[10][2][3]+[10](1-[2])(1-[3])+(1-[10])(1-[2])"	--
01	[%, %, %, %, %, %,%,%,%, %, 0]	10	"(0)[2][3]+(0)(1-[2])(1-[3])+(1-(0))(1-[2])"	"[10][2][3]+[10](1-[2])(1-[3])+(1-[10])(1-[2])"	--
02	[%, %, %, %, %, %,%,%,%, %, 1]	10	"(0)[2][3]+(0)(1-[2])(1-[3])+(1-(0))(1-[2])"	"[10][2][3]+[10](1-[2])(1-[3])+(1-[10])(1-[2])"	--
03	[%, %, %, %, %, %,%,%,%, %, 1]	10	"(0)[2][3]+(0)(1-[2])(1-[3])+(1-(0))(1-[2])"	"(1)[2][3]+(1)(1-[2])(1-[3])+(1-(1))(1-[2])"	--
04	[%, %, %, %, %, %,%,%,%, %, 1]	10	"(0)[2][3]+(0)(1-[2])(1-[3])+(1-(0))(1-[2])"	"(1)[2][3]+(1)(1-[2])(1-[3])+(1-(1))(1-[2])"	"((0)[2][3]+(0)(1-[2])(1-[3])+(1-(0))(1-[2]))((1)[2][3]+(1)(1-[2])(1-[3])+(1-(1))(1-[2]))"
05	[%, %, %, %, %, %,%,%,%, %, 1]	10	"(0)[2][3]+(0)(1-[2])(1-[3])+(1-(0))(1-[2])"	"(1)[2][3]+(1)(1-[2])(1-[3])+(1-(1))(1-[2])"	"((0)[2][3]+(0)(1-[2])(1-[3])+(1-(0))(1-[2]))((1)[2][3]+(1)(1-[2])(1-[3])+(1-(1))(1-[2]))"

Expected	Actual	Result
productOfElimination = "(1 - [2])(1 - [3])"	productOfElimination = "((0)[2][3]+(0)(1-[2])(1-[3])+(1-(0))(1-[2]))((1)[2][3]+(1)(1-[2])(1-[3])+(1-(1))(1-[2]))"	???

There is no way for a tester to quickly be able to look upon ((0)[2][3]+(0)(1-[2])(1-[3])+(1-(0))(1-[2]))((1)[2][3]+(1)(1-[2])(1-[3])+(1-(1))(1-[2])) and realise hastily "oh -- of course -- that simplifies to (1 - [2])(1 - [3])".

This renders observers cognisant to the fact that automated tests, when they are written, will need to be able to tell whether two functions of type delta are equal. The quickest way that comes to mind here is to subtract the two functions, then develop with respect to all of their symbols, and assert that the development is naught:

$$G(\delta_1 - \delta_2, \varphi(\delta_1 - \delta_2)) = 0;$$

Performing such an evaluation (with a little help from JavaScript to evaluate $[\delta_1 - \delta_2](\theta_2, \theta_3)$) yields that the equations are in parity:

If the following condition holds, then the equations are equal:
$G(\delta_1 - \delta_2, \varphi(\delta_1 - \delta_2)) = 0;$

LHS:
$G(\delta_1 - \delta_2, \varphi(\delta_1 - \delta_2));$
$G(\delta_1 - \delta_2, \{\theta_2, \theta_3\});$
$(1 - \theta_2)(1 - \theta_3)[\delta_1 - \delta_2](0, 0) + (1 - \theta_2)\theta_3[\delta_1 - \delta_2](0, 1) + \theta_2(1 - \theta_3)[\delta_1 - \delta_2](1, 0) + \theta_2\theta_3[\delta_1 - \delta_2](1, 1);$

Which is only reducible to naught under the condition:

$|[\delta_1 - \delta_2](0, 0)| + |[\delta_1 - \delta_2](0, 1)| + |[\delta_1 - \delta_2](1, 0)| + |[\delta_1 - \delta_2](1, 1)|;$
$|0| + |0| + |0| + |0|;$
$0;$

= RHS;

$\therefore G(\delta_1 - \delta_2, \varphi(\delta_1 - \delta_2)) = 0$, which is the condition of equality.

The JavaScript used follows:

```
function deltaSubtraction(two, three)
{
    return ((0)*(two)*(three)+(0)*(1-(two))*(1-(three))+(1-(0))*(1-(two)))
    *((1)*(two)*(three)+(1)*(1-(two))*(1-(three))+(1-(1))*(1-(two)))-(1-
    (two))*(1-(three));
}
document.writeln(deltaSubtraction(0, 0));
document.writeln(deltaSubtraction(0, 1));
document.writeln(deltaSubtraction(1, 0));
document.writeln(deltaSubtraction(1, 1));
```

1.19. BOOL RELATION::EVALUATE(VOID)

This module calls what is assumed to be an inbuilt library or source routine in order to evaluate a string, and thus, all of the testable logic is migrated. In the appendix (`VII.), a sample implementation is attached along with low level unit tests, and several test cases of the system wholly. This sufficiently details the logic of the this routine.

1.20. INT[] RELATION::GET_INVOLVED_SYMBOL_LIST(VOID)

This function is one that is very much dependent on string manipulation functions, and thus, manual deskchecking would be quite obfuscated in aspect. Instead, a shoddy JavaScript emulation can be used:

```
function getInvolvedSymbolList(data)
{
    var symbolList = [];
```

```
    var symbolOccurrences = data.split('[');
    var numOccurrences = symbolOccurrences.length;
    for (var occurrence = 0; occurrence < numOccurrences; ++occurrence)
    {
        var currentSymbolString = symbolOccurrences[occurrence];
        currentSymbolString = currentSymbolString.split(']')[0];
        var currentSymbol = eval(currentSymbolString);
        if (('.' + symbolList.join('.') + '.').search('.' + currentSymbol +
            '.') == -1)
        {
            symbolList[symbolList.length] = currentSymbol;
        }
    }
    return symbolList;
}
```

Executing this with any test case return what is clearly an error -- which is isolated to the first index, for example:

```
getInvolvedSymbolList("12*3[1] + [1] + [2]-23*12[13]+[1]-[12]");
```

Returns:

```
[36, 1, 2, 13, 12]
```

Whereby 36 is not a symbol, but, it can be quickly inferred that 12*3 = 36, and that there is one more symbol than would be expected. A quick glance of the algorithm with this knowledge, which justifies the expectation of some sort of boundary error, yields that the for loop is executing one too many times, hence the one extra symbol; and that this is the first case which is ran erroneously, thus, the appearance of 32 at both the beginning of the array, and at the beginning of the string. The correction simply involves the variable *occurrence* to be allocated one value higher, thus, the beginning of the for loop becomes:

```
for (var occurrence = 1; occurrence < numOccurrences; ++occurrence)
```

It can secondly be noticed on the execution of a case such as:

```
getInvolvedSymbolList("[0] + (1 - [0])");
```

An array is returned:

```
[0]
```

Yet,

$$G(\theta + (1 - \theta), \{\theta\}) = 1$$

This behaviour is in contradiction to the majority of planned test cases, but is not detrimental to the functionality of the subroutine, although it is detrimental to the efficiency. Generating new test cases and results with the emulation:

```
function equal(a, b)
{
    if (a.length != b.length)
    {
        return false;
    }
    var holdsHereto = true;
    for (var iii = 0; iii < a.length; ++iii)
    {
        if (a[iii] != b[iii])
        {
            holdsHereto = false;
        }
    }
    return holdsHereto;
}

function testWith(arg, expected)
{
    var state = (equal(getInvolvedSymbolList(arg), expected)) ? "correct" :
"incorrect";
    document.write("getInvolvedSymbolList(" + arg + ") -> " + state +
"<br>");
}
testWith("1", []);
testWith("[1]", [1]);
testWith("0[1]", [1]);
testWith("1[1] - [1]", [1]);
testWith("12 + 23*2 - 1 + [9000] - 3[730]", [9000, 730]);
testWith("[1] + [2] + [3] + [1] + [5]", [1, 2, 3, 5]);
testWith(") -- (this test is intentionally erroneous", [12]);
```

Executing these tests:

```
getInvolvedSymbolList(1) -> correct
getInvolvedSymbolList([1]) -> correct
getInvolvedSymbolList(0[1]) -> correct
getInvolvedSymbolList(1[1] - [1]) -> correct
getInvolvedSymbolList(12 + 23*2 - 1 + [9000] - 3[730]) -> correct
getInvolvedSymbolList([1] + [2] + [3] + [1] + [5]) -> correct
getInvolvedSymbolList() -- (this test is intentionally erroneous) ->
    incorrect
```

The results can be manually examined if the *equal* function is not trusted:

```
getInvolvedSymbolList(1) -> [] vs []
getInvolvedSymbolList([1]) -> [1] vs [1]
getInvolvedSymbolList(0[1]) -> [1] vs [1]
```

```
getInvolvedSymbolList(1[1] - [1]) -> [1] vs [1]
getInvolvedSymbolList(12 + 23*2 - 1 + [9000] - 3[730]) -> [9000, 730] vs
    [9000, 730]
getInvolvedSymbolList([1] + [2] + [3] + [1] + [5]) -> [1, 2, 3, 5] vs
    [1, 2, 3, 5]
getInvolvedSymbolList() -- (this test is intentionally erroneous) -> [] vs
    [12]
```

1.21. VOID RELATION::INSERT_ARGUMENTS(TRIAD[] SYMBOL_VALUES)

The algorithm for this module can be tested as a few general cases from the planned tests:

```
00 FOR symbol = 0 TO numSymbols
01     currentSymbolValue = symbolValues[symbol]
02     IF currentSymbolValue is not in the quantum state 0/0 THEN
03         searchTerm = '[', symbol, ']'
04         replacementTerm = '(', currentSymbolValue, ')'
05         Replace occurrences of searchTerm in this.data with
               replacementTerm
06     ENDIF
07 NEXT symbol
08
```

Relation::insertArguments(['v', 'v', 'v', 'v', '1', 'v', '1']) on "[4][0][1]+(1-[0])[1](1-[4])"							
Line	symbolValues	symbol	numSymbols	currentSymbolValue	searchTerm	replacementTerm	this.data
00	['v', 'v', 'v', 'v', '1', 'v', '1']	0	7	--	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
01	['v', 'v', 'v', 'v', '1', 'v', '1']	0	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
02	['v', 'v', 'v', 'v', '1', 'v', '1']	0	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
07	['v', 'v', 'v', 'v', '1', 'v', '1']	1	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
00	['v', 'v', 'v', 'v', '1', 'v', '1']	1	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
01	['v', 'v', 'v', 'v', '1', 'v', '1']	1	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
02	['v', 'v', 'v', 'v', '1', 'v', '1']	1	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
07	['v', 'v', 'v', 'v', '1', 'v', '1']	2	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
00	['v', 'v', 'v', 'v', '1', 'v', '1']	2	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
01	['v', 'v', 'v', 'v', '1', 'v', '1']	2	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
02	['v', 'v', 'v', 'v', '1', 'v', '1']	2	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
07	['v', 'v', 'v', 'v', '1', 'v', '1']	3	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
00	['v', 'v', 'v',	3	7	'v'	--	--	"[4][0][1]+(1-

	'v', '1', 'v', '1']						[0])[1](1-[4])"
01	['v', 'v', 'v', 'v', '1', 'v', '1']	3	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
02	['v', 'v', 'v', 'v', '1', 'v', '1']	3	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
07	['v', 'v', 'v', 'v', '1', 'v', '1']	4	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
00	['v', 'v', 'v', 'v', '1', 'v', '1']	4	7	'v'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
01	['v', 'v', 'v', 'v', '1', 'v', '1']	4	7	'1'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
02	['v', 'v', 'v', 'v', '1', 'v', '1']	4	7	'1'	--	--	"[4][0][1]+(1-[0])[1](1-[4])"
03	['v', 'v', 'v', 'v', '1', 'v', '1']	4	7	'1'	"[4]"	--	"[4][0][1]+(1-[0])[1](1-[4])"
04	['v', 'v', 'v', 'v', '1', 'v', '1']	4	7	'1'	"[4]"	"(1)"	"[4][0][1]+(1-[0])[1](1-[4])"
05	['v', 'v', 'v', 'v', '1', 'v', '1']	4	7	'1'	"[4]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
07	['v', 'v', 'v', 'v', '1', 'v', '1']	5	7	'1'	"[4]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
00	['v', 'v', 'v', 'v', '1', 'v', '1']	5	7	'1'	"[4]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
01	['v', 'v', 'v', 'v', '1', 'v', '1']	5	7	'v'	"[4]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
02	['v', 'v', 'v', 'v', '1', 'v', '1']	5	7	'v'	"[4]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
07	['v', 'v', 'v', 'v', '1', 'v', '1']	6	7	'v'	"[4]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
00	['v', 'v', 'v', 'v', '1', 'v', '1']	6	7	'v'	"[4]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
01	['v', 'v', 'v', 'v', '1', 'v', '1']	6	7	'1'	"[4]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
02	['v', 'v', 'v', 'v', '1', 'v', '1']	6	7	'1'	"[4]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
03	['v', 'v', 'v', 'v', '1', 'v', '1']	6	7	'1'	"[6]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
04	['v', 'v', 'v', 'v', '1', 'v', '1']	6	7	'1'	"[6]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
05	['v', 'v', 'v', 'v', '1', 'v', '1']	6	7	'1'	"[6]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
07	['v', 'v', 'v', 'v', '1', 'v', '1']	7	7	'1'	"[6]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
00	['v', 'v', 'v', 'v', '1', 'v', '1']	7	7	'1'	"[6]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"
08	['v', 'v', 'v', 'v', '1', 'v', '1']	7	7	'1'	"[6]"	"(1)"	"(1)[0][1]+(1-[0])[1](1-(1))"

Expected	Actual	Result
this.data = "(1)[0][1] + (1 - [0])[1](1 - (1))"	this.data = "(1)[0][1]+(1-[0])[1](1-(1))"	Success

Relation::insertArguments([]) on "(1)[0][1]+(1-[0])[1](1-(1))"							
Line	symbolValues	symbol	numSymbols	currentSymbolValue	searchTerm	replacementTerm	this.data
00	['v', 'v', 'v', 'v', '1', 'v', '1']	0	0	--	--	--	"(1)[0][1]+(1-[0])[1](1-(1))"
08	['v', 'v', 'v', 'v', '1', 'v', '1']	0	0	--	--	--	"(1)[0][1]+(1-[0])[1](1-(1))"

Expected	Actual	Result
this.data = "(1)[0][1] + (1 - [0])[1](1 - (1))"	this.data = "(1)[0][1]+(1-[0])[1](1-(1))"	Success

Walking through the algorithm, it becomes obvious that these results are generalisable.

1.22. RELATION::RELATION(STRING CONSTRUCTOR)

This algorithm simply allocates a member variable for a given string argument, although, in the actual implementation, may be extended to validate the constructor, perform perhaps a transposition, and eliminate quantum states (although special design consideration must be had for entangled particles). To furnish the correctness of this routine, a general case run-through can still be performed:

```
00 this.data = constructor
01
```

Relation::Relation(<relation>)		
Line	constructor	this.data
00	<relation>	<relation>
01	<relation>	<relation>

Expected	Actual	Result
this.data = <relation>	this.data = <relation>	Success

2. DEBUGGING

2.0. NOTES

In a project of moderate complexity -- but in any project generally -- bugs are a natural part of development. Small syntactical mistakes are made and iteratively rectified without much thought. Logic errors, whether they cause runtime crashes or not, inherently will exist, and mostly (due primarily to the dedicated planning stages and an understanding of the system wherefore), their sources can be thought through in a number of seconds, and a fix can be derived in little more time.

Occasionally, a more difficult class of errors arrive. This may see the need of more concentrated thought, or the haphazard prodding of the program so that your model of the system models what is actually present in the code. Partial restructures of subroutines may be requisite once the error is exposed, but verily, the documentation of such mild inconveniences can hardly be feigned as necessary. Very rarely though, the infinitely sceptical programmer will be surprised at the behaviour of a system -- and this is a very special case.

Programmers ought to expect to be surprised, for that is the nature of systems. It should be expected that a system doesn't work, and surprising behaviour should itself be no real surprise. There must be observed harsh scepticism, and all unfounded hope suppressed. "Why doesn't this work" becomes an invalid question, replaced with "for which reason does this not work", and this is where the surprise can be found. Debuggers can now, by more comfortably dialectical projections, conjure the basic framework of thought: "Things aren't working very well. By my memory, I foresore a dozen possible causes for erroneous behaviour and made attempts to fix them. I wonder which of my faulty fixes produced this error, or rather if my fixes actually worked, which would be quite the oddity, my wonder lies in which of the unknown problems are encountered in this circumstance."

For these more unanticipated circumstances, there are a wide range of methods and tools available to help identify and fix the bug. Some arbitrarily selected instances follow of errors that were surprising, but many more will surely pass by without documentation by necessity of brevity, or because their discussion is confined to more informal areas such as logbooks. This section serves to typify the existence of debugging practices.

2.1. INSTANCE 1: PYTHON DIRECTORIES

```
# testRelation.py
# Nicholas Killeen,
# 12th June 2016.
# Unit tests for Relation.py.

import Relation;

def testRelation():
    x = Relation.Relation("okay");
    assert(False);
```

```
# Relation.py
# Nicholas Killeen,
# 12th June 2016.
# Class to store and algebraically manipulate Boolean relations.

class Relation:
    def __init__(this, msg):
        print("i am being constructed " + msg);
```

Despite both of these modules being in the same directory, the program terminates:

```
Testing Relation ... Traceback (most recent call last):
File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\testPLE.py", line 25, in <module>
testPLE();
File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\testPLE.py", line 19, in testPLE
testRelation.testRelation();
File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\Relation\testRelation.py", line 9, in testRelation
x = Relation.Relation("okay");
AttributeError: 'module' object has no attribute 'Relation'
Press any key to continue . . .
```

The error lies in file sharing rather than the incorrect definition of a class, as verified by the correct output being produced in the Python interpreter, below:

```
>>> class ArbClass2:
      def __init__(this, msg):
            print("hello from ArbClass2, {0}".format(msg));

>>> y = ArbClass2("i am receiving you");
hello from ArbClass2, i am receiving you
>>>
```

It next comes to attention that, perhaps, the error is due to a naming collision, but the following modifications still produce the same error:

```
# testRelation.py
# Nicholas Killeen,
# 12th June 2016.
# Unit tests for Relation.py.

import Relation;

def testRelation():
    x = Relation.Relatiohno("okay");
    assert(False);
```

```
# Relation.py
# Nicholas Killeen,
# 12th June 2016.
# Class to store and algebraically manipulate Boolean relations.

class Relatiohno:
    def __init__(this, msg):
        print("i am being constructed {0}".format(msg));
```

It is next instinct that Python, in its infinitely obtuse aspect, cannot actually find the Relation module, and has in fact been trying to communicate this is: "AttributeError: 'module' object has no attribute 'Relation'". The error isn't that the attribute isn't found, it is that the module 'Relation' cannot be found. After recognising this, rectification can easily be made:

```
import Relation.Relation as Relation;
```

After which, the program correctly produces the sought assertion error:

```
Testing Relation ... i am being constructed okay
Traceback (most recent call last):
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\testPLE.py", line 25, in <module>
    testPLE();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\testPLE.py", line 19, in testPLE
    testRelation.testRelation();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\Relation\testRelation.py", line 10, in testRelation
    assert(False);
AssertionError
Press any key to continue . . .
```

2.2. INSTANCE 2: QUANTUM_COUNTER OVERFLOW

The partial source for QuantumCounter produces unexpected behaviour, observed early into the unit testing process:

```
# Increment the current state of the counter, moving over quantum
# bits. Overflow resets the counter.
def increment(this):
    bitNumber = len(this.bitStates) - 1;
    carry = 1;
    # iterate over each bit, right to left, stop if there is no
    # longer a bit to carry
    while (bitNumber >= 0 and carry == 1):
        currentState = this.bitStates[bitNumber];
        if (currentState == 0):
            this.bitStates[bitNumber] = 1;
            carry = 0;
        elif (currentState == 1):
            this.bitStates[bitNumber] = 0;
```

The tests:

```
def testQuantumCounter():
    # test constructor __init__, read, and increment
   quantumCounter1 = QuantumCounter.QuantumCounter([0]);
   assert(quantumCounter1.read() == [0]);
   quantumCounter1.increment();
   assert(quantumCounter1.read() == [1]);
   quantumCounter1.increment();
   assert(quantumCounter1.read() == [0]);
```

Produce the error:

```
Testing QuantumCounter ... Traceback (most recent call last):
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\testPLE.py", line 25, in <module>
    testPLE();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\testPLE.py", line 16, in testPLE
    testQuantumCounter.testQuantumCounter();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\QuantumCounter\testQuantumCounter.py", line 16, in testQuantum
Counter
    assert(quantumCounter1.read() == [0]);
AssertionError
Press any key to continue . . .
```

Specifically, the error can be systematically reduced to its smallest possible form:

```
>>> notGood = QuantumCounter([0]);
>>> notGood.increment();
>>> notGood.increment();
>>> notGood.read();
[1]
>>>
```

It is clear that the increment function doesn't correctly toggle the states [1] and [2] on incrementation, but, the addition of debugging output statements with the aid of the Visual Studio interpreter "send to interactive" command, means that the class's behaviour can be analysed in a controlled environment. The beginning of the debugging output statement looks as follows:

```
# Increment the current state of the counter, moving over quantum
# bits. Overflow resets the counter.
def increment(this):
    bitNumber = len(this.bitStates) - 1;
    carry = 1;
    # iterate over each bit, right to left, stop if there is no
    # longer a bit to carry
    print("thinking about iterating ...");
    while (bitNumber >= 0 and carry == 1):
        print("    iterating");
        print("    the current bitState is -- ");
```

```
        if (currentState == 0):
            this.bitStates[bitNumber] = 1;
            carry = 0;
        elif (currentState == 1):
            this.bitStates[bitNumber] = 0;
        print("    thinking about iterating ...");
    print("i didn't iterate\nterminating ...");
```

The incompleteness of the annotation follows from a drastic realisation. Despite the program "thinking about iterating", and "iterating", it, in fact, "didn't iterate", and it is a surprise that the program terminated at all. The problem is that bitNumber never gets decremented, so the method only looks at the final bit in the array, and toggles it in such a way that it will always end up storing 1 (for that is when the carry bit will be equal to 0, and the loop will terminate). Addition of a decrement operator on bitNumber gives success to the most basic unit tests:

```
Testing QuantumCounter ... passed!
```

2.3. INSTANCE 3: METHODS VERSUS FUNCTIONS

The following code, when ran with the appended black box unit test, produces an error (also appended):

```
def getFactAggregation(this):
    # optimise each fact
    optimisedFactList = [];
    for fact in this.factList:
        optimisedFact = fact.optimise();
        optimisedFactList.append(optimisedFact);

    UNITY = Relation.Relation('1');
    aggregate = Relation.Relation('0');
    factNumber = 0;
    numFacts = len(optimisedFactList);
    while (factNumber < numFacts):
        priorFactNumber = 0;
        productOfPriorFacts = Relation.Relation('1');
        while (priorFactNumber < factNumber):
            productOfPriorFacts = productOfPriorFacts * (UNITY -
                optimisedFactList[priorFactNumber]);
            priorFactNumber += 1;
        currentFact = optimisedFactList[factNumber];
        aggregate = aggregate + currentFact * productOfPriorFacts;
        factNumber += 1;
    return aggregate;
```

```
# test getFactAggregation
factDatabase3 = FactDatabase.FactDatabase();
relation11 = Relation.Relation("[0]");
factDatabase3.appendFact(relation11);
assert(factDatabase3.getFactAggregation() == relation11);
```

```
Testing FactDatabase ....... Traceback (most recent call last):
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\__init__.py", line 7, in <module>
    main();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\__init__.py", line 6, in main
    testPLE.testPLE();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\testPLE.py", line 26, in testPLE
    testFactDatabase.testFactDatabase();
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\FactDatabase\testFactDatabase.py", line 98, in testFactDatabas
e
    assert(factDatabase3.getFactAggregation() == relation11);
  File "C:\Users\Nicholas\Documents\Dir\Programming\Python\Projects\4 -
P.L.E\PLE\FactDatabase\FactDatabase.py", line 51, in getFactAggregation
    aggregate = aggregate + currentFact * productOfPriorFacts;
TypeError: unsupported operand type(s) for *: 'NoneType' and 'Relation'
Press any key to continue . . .
```

For this particular type of error, Python has it made known what the last line was before the crash, so there needn't be any manual location of the last functional line through single line stepping or printing debugging output statements. With this known, it is next appropriate to examine the contents of the variables at the line before the crash, which PTVS allows to be done with ease through breakpoints, or, in the context menu, a "run to cursor" command. In this debugging mode, Visual Studio allows for variable names to be hovered over in the code so as to display their contents It can then be found that the variable currentFact is of type NoneType, thus producing the error, and this behaviour can be traced back with mild investigative work. The None value can be traced back to the highlighted line:

```
def getFactAggregation(this):
    # optimise each fact
    optimisedFactList = [];
    for fact in this.factList:
        optimisedFact = fact.optimise();
        optimisedFactList.append(optimisedFact);
```

It can be brought to memory that fact.optimise is a method, not a function, namely, it does not return anything, and merely modifies the object which the method was called on. This is the obvious source of the error. Correction sees this portion of the routine modified:

```
def getFactAggregation(this):
    # optimise each fact
```

```
    optimisedFactList = [];
    for fact in this.factList:
        optimisedFact = Relation.Relation();
        optimisedFact.copyFrom(fact);
        optimisedFact.optimise();
        optimisedFactList.append(optimisedFact);
```

And finally, wholly:

```
Testing Fraction .......... passed!
Testing QuantumCounter ..... passed!
Testing Relation .......... passed!
Testing FactDatabase ....... passed!

All tests passed!!!
You are awesome!
Press any key to continue . . .
```

3. UNIT TESTING

3.0. NOTES

In strict accordance with the test plan (III. 3.4.), unit tests accompany all Classes. This allows actual and expected output for each method to be automatically tested against a suite of test cases, meaning that every time the program is compiled (or linked and then interpreted in Python's strange case), the tests will run without any conscious effort. This makes it obvious when a change is made that breaks an otherwise working function or method, thus giving courage to refactor. Not all components of the system, sadly, are able to be automatically tested -- namely, it will be found that the CLI and GUI source files lack accompanying unit tests. The responsibility of correctness in this area falls to V. 4..

3.1. TEST_FACT_DATABASE.PY

```
# FactDatabase/testFactDatabase.py
# Nicholas Killeen,
# 19th June 2016.
# Unit tests for FactDatabase.py.

import FactDatabase.FactDatabase as FactDatabase;
import Relation.Relation as Relation;

def testFactDatabase():
    # test constructor __init__, appendFact, and getFactList
    factDatabase1 = FactDatabase.FactDatabase();
    assert(factDatabase1.getFactList() == []);
    relation1 = Relation.Relation("[0]");
    successStateAdd1 = factDatabase1.appendFact(relation1);
    assert(successStateAdd1 == True);
    assert(factDatabase1.getFactList()[0] == relation1);
    assert(len(factDatabase1.getFactList()) == 1);

    relation2 = Relation.Relation("[0] + 1");
    successStateAdd2 = factDatabase1.appendFact(relation2);
    assert(successStateAdd2 == False);
    assert(factDatabase1.getFactList()[0] == relation1);
    assert(len(factDatabase1.getFactList()) == 1);

    relation3 = Relation.Relation("[0] + (1 - [0])[12]");
    successStateAdd3 = factDatabase1.appendFact(relation3);
    assert(successStateAdd3 == True);
    assert(factDatabase1.getFactList()[0] == relation1);
    assert(factDatabase1.getFactList()[1] == relation3);
    assert(len(factDatabase1.getFactList()) == 2);

    successStateAdd4 = factDatabase1.appendFact("pulled pork");
    assert(successStateAdd4 == False);
    assert(factDatabase1.getFactList()[0] == relation1);
    assert(factDatabase1.getFactList()[1] == relation3);
    assert(len(factDatabase1.getFactList()) == 2);
```

```
    relation5 = Relation.Relation("[12](1 - [5]) + [5](1 - [12])[0]");
    successStateAdd5 = factDatabase1.appendFact(relation5);
    assert(successStateAdd5 == True);
    assert(factDatabase1.getFactList()[0] == relation1);
    assert(factDatabase1.getFactList()[1] == relation3);
    assert(factDatabase1.getFactList()[2] == relation5);
    assert(len(factDatabase1.getFactList()) == 3);

    # test deleteFact
    factDatabase2 = FactDatabase.FactDatabase();
    successStateDelete1 = factDatabase2.deleteFact(0);
    assert(successStateDelete1 == False);
    successStateDelete2 = factDatabase2.deleteFact(1);
    assert(successStateDelete2 == False);
    successStateDelete3 = factDatabase2.deleteFact(-1);
    assert(successStateDelete3 == False);

    relation6 = Relation.Relation("[1][2](1 - [3])");
    factDatabase2.appendFact(relation6);
    successStateDelete4 = factDatabase2.deleteFact(0);
    assert(successStateDelete4 == True);
    assert(len(factDatabase2.getFactList()) == 0);

    relation7 = Relation.Relation("[1][8] + (1 - [1])(1 - [8])");
    factDatabase2.appendFact(relation7);
    relation8 = Relation.Relation("[1][2] + [2](1 - [8])(1 - [1])");
    factDatabase2.appendFact(relation8);
    relation9 = Relation.Relation("[2](1 - [3]) + [4][3](1 - [2])");
    factDatabase2.appendFact(relation9);
    relation10 = Relation.Relation("0");
    factDatabase2.appendFact(relation10);
    successStateDelete5 = factDatabase2.deleteFact(4);
    assert(successStateDelete5 == False);
    successStateDelete6 = factDatabase2.deleteFact(0);
    assert(successStateDelete6 == True);
    factList1 = factDatabase2.getFactList();
    assert(factList1[0] == relation8);
    assert(factList1[1] == relation9);
    assert(factList1[2] == relation10);
    assert(len(factList1) == 3);

    successStateDelete7 = factDatabase2.deleteFact(1);
    assert(successStateDelete7 == True);
    factList2 = factDatabase2.getFactList();
    assert(factList2[0] == relation8);
    assert(factList2[1] == relation10);
    assert(len(factList2) == 2);

    successStateDelete8 = factDatabase2.deleteFact(1);
    assert(successStateDelete8 == True);
    factList3 = factDatabase2.getFactList();
    assert(factList3[0] == relation8);
```

```
    assert(len(factList3) == 1);

    # test getFactAggregation
    factDatabase3 = FactDatabase.FactDatabase();
    relation11 = Relation.Relation("[0]");
    factDatabase3.appendFact(relation11);
    assert(factDatabase3.getFactAggregation() == relation11);
    relation12 = Relation.Relation("[1]");
    factDatabase3.appendFact(relation12);
    relation13 = Relation.Relation("[1] + [0](1 - [1])");
    assert(factDatabase3.getFactAggregation() == relation13);
    relation14 = Relation.Relation("(1 - [2][3])");
    factDatabase3.appendFact(relation14);
    relation15 = Relation.Relation("[0] + [1](1 - [0]) + (1 - [2][3])(1"
        " - [1])(1 - [0])");
    assert(factDatabase3.getFactAggregation() == relation15);
    relation16 = Relation.Relation("[3] + (1 - [3])(1 - [2])");
    factDatabase3.appendFact(relation16);
    relation17 = Relation.Relation("[0] + [1](1 - [0]) + (1 - [1])(1 - "
        "[0])");
    assert(factDatabase3.getFactAggregation() == relation17);

    factDatabase4 = FactDatabase.FactDatabase();
    relation18 = Relation.Relation("0");
    assert(factDatabase4.getFactAggregation() == relation18);
```

3.2. TEST_FRACTION.PY

```
# Fraction/testFraction.py
# Nicholas Killeen,
# 19th June 2016.
# Unit tests for Fraction.py.

import Fraction.Fraction as Fraction;

def testFraction():
    # test constructor __init__, getNumerator, and getDenominator
    fraction1 = Fraction.Fraction("chopped capsicum", "diced tomato");
    assert(fraction1.getNumerator() == "chopped capsicum");
    assert(fraction1.getDenominator() == "diced tomato");

    fraction2 = Fraction.Fraction("shredded chicken",
        "shredded chicken");
    assert(fraction2.getNumerator() == "shredded chicken");
    assert(fraction2.getDenominator() == "shredded chicken");
```

3.3. TEST_INTERFACE.PY

```
# Interface/testInterface.py
# Nicholas Killeen,
# 19th June 2016.
# Unit tests for Interface.py.
```

```
import Interface.Interface as Interface;
import Relation.Relation as Relation;

def testInterface():
    # test constructor __init__, getFactList, addFact, and deleteFact
    errorList = [];
    interface1 = Interface.Interface(errorList.append);
    assert(interface1.getFactList() == []);

    relation1 = Relation.Relation("1");
    interface1.addFact(relation1);
    assert(interface1.getFactList() == [relation1]);

    relation2 = Relation.Relation("0");
    interface1.addFact(relation2);
    assert(interface1.getFactList() == [relation1, relation2]);

    relation3 = Relation.Relation("[0]");
    interface1.addFact(relation3);
    assert(interface1.getFactList() == [relation1, relation2,
        relation3]);

    assert(len(errorList) == 0);
    interface1.deleteFact(-1);
    assert(interface1.getFactList() == [relation1, relation2,
        relation3]);
    assert(len(errorList) == 1);

    interface1.deleteFact(0);
    assert(interface1.getFactList() == [relation2, relation3]);
    interface1.deleteFact(1);
    assert(interface1.getFactList() == [relation2]);

    interface1.deleteFact(1);
    assert(interface1.getFactList() == [relation2]);
    assert(len(errorList) == 2);

    relation4 = Relation.Relation("2");
    interface1.addFact(relation4);
    assert(interface1.getFactList() == [relation2]);
    assert(len(errorList) == 3);

    # testing setRequest, and getRequest
    interface1.setRequest(relation4);
    assert(len(errorList) == 4);
    request = interface1.getRequest();
    assert(request == None);
    assert(len(errorList) == 5);

    relation5 = Relation.Relation("[1]");
    interface1.setRequest(relation5);
    request = interface1.getRequest();
    assert(request == relation5);
```

```
    assert(len(errorList) == 5);

    errorList.clear();

    # testing generateProcessor, and retrieveSoughtInformation
    interface2 = Interface.Interface(errorList.append);
    interface2.retrieveResponse();
    assert(len(errorList) == 1);
    interface2.generateProcessor();
    assert(len(errorList) == 2);

    interface2.addFact(Relation.Relation("1"));
    interface2.setRequest(Relation.Relation("1"));
    interface2.generateProcessor();
    assert(len(errorList) == 2);
    interface2.generateProcessor();
    assert(len(errorList) == 2);

    errorList.clear();

    interface3 = Interface.Interface(errorList.append);
    handler = interface3.generateProcessor();
    assert(handler == None);
    assert(len(errorList) == 1);

    interface3.addFact(Relation.Relation("[1]"));
    interface3.setRequest(Relation.Relation("[1]"));
    handler = interface3.generateProcessor();
    assert(len(errorList) == 1);
    assert(handler != None);

    interface3.retrieveResponse();
    assert(len(errorList) == 2);
    handler();
    interface3.retrieveResponse();

    quaesitum = interface3.retrieveResponse();
    assert(len(errorList) == 2);
    assert(quaesitum.getNumerator().evaluate() / quaesitum.
        getDenominator().evaluate() == 0);

    errorList.clear();

    interface3 = Interface.Interface(errorList.append);
    interface3.addFact(Relation.Relation("[1](1 - [0])"));
    interface3.addFact(Relation.Relation("[0](1 - [1])"));
    interface3.setRequest(Relation.Relation("[1]"));
    quaesitum = interface3.retrieveResponse();
    assert(len(errorList) == 1);
    handler = interface3.generateProcessor();
    quaesitum = interface3.retrieveResponse();
    assert(quaesitum == None);
    assert(len(errorList) == 2);
```

```
    handler();
    quaesitum = interface3.retrieveResponse();
    assert(quaesitum != None);
    assert(len(errorList) == 2);
```

3.4. TEST_PROCESSOR.PY

```
# Processor/testProcessor.py
# Nicholas Killeen,
# 19th June 2016.
# Unit tests for Processor.py.

import Processor.Processor as Processor;
import Relation.Relation as Relation;
import QuantumCounter.QuantumCounter as QuantumCounter;

def testProcessor():
    # test all methods: __init__, go, hasFinished, getProductWith0 and
    # getProductWith1.

    # group 1
    aggregate1 = Relation.Relation("[0](1 - [1]) + [1](1 - [0])");
    soughtDevelopment1 = Relation.Relation("[0]");

    quantumCounter1 = QuantumCounter.QuantumCounter();
    quantumCounter1.createCoefficientMap(soughtDevelopment1.
        getSymbolList());
    processor1 = Processor.Processor(aggregate1, soughtDevelopment1,
        quantumCounter1, 2);
    processor1.go();
    assert(processor1.getProductWith0() == Relation.Relation("[1]"));
    assert(processor1.getProductWith1() == Relation.Relation("1-[1]"));

    # group 2
    aggregate2_3 = Relation.Relation("[0](1 - [1])[4] + (1 - [4])[5][1]"
        "(1 - [0])");
    soughtDevelopment2_3 = Relation.Relation("[0][1](1 - [5])");

    quantumCounter2 = QuantumCounter.QuantumCounter();
    quantumCounter2.createCoefficientMap(soughtDevelopment2_3.
        getSymbolList());
    processor2 = Processor.Processor(aggregate2_3,
        soughtDevelopment2_3, quantumCounter2, 5);
    assert(processor2.hasFinished() == False);
    processor2.go();
    assert(processor2.hasFinished() == True);

    quantumCounter3 = QuantumCounter.QuantumCounter();
    quantumCounter3.createCoefficientMap(soughtDevelopment2_3.
        getSymbolList());
    quantumCounter3.add(5);
    processor3 = Processor.Processor(aggregate2_3,
        soughtDevelopment2_3, quantumCounter3, 3);
```

```
    assert(processor3.hasFinished() == False);
    processor3.go();
    assert(processor3.hasFinished() == True);

    productWith0For2_3 = (processor2.getProductWith0() *
        processor3.getProductWith0());
    assert(productWith0For2_3 == Relation.Relation("0"));
    productWith1For2_3 = (processor2.getProductWith1() *
        processor3.getProductWith1());
    assert(productWith1For2_3 == Relation.Relation("0"));

    # group 3
    aggregate4_5 = Relation.Relation("[2](1 - [3]) + (1 - [2])([3][4] +"
        "(1 - [3])(1 - [4]))");
    soughtDevelopment4_5 = Relation.Relation("[2](1 - [3])[5] + (1 - [5"
        "])[2](1 - [3]) + [3](1 - [2])");

    quantumCounter4 = QuantumCounter.QuantumCounter();
    quantumCounter4.createCoefficientMap(soughtDevelopment4_5.
        getSymbolList());
    processor4 = Processor.Processor(aggregate4_5,
        soughtDevelopment4_5, quantumCounter4, 4);
    processor4.go();

    quantumCounter5 = QuantumCounter.QuantumCounter();
    quantumCounter5.createCoefficientMap(soughtDevelopment4_5.
        getSymbolList());
    quantumCounter5.add(4);
    processor5 = Processor.Processor(aggregate4_5,
        soughtDevelopment4_5, quantumCounter5, 4);
    processor5.go();

    productWith0For4_5 = (processor4.getProductWith0() *
        processor5.getProductWith0());
    assert(productWith0For4_5 == Relation.Relation("0"));
    productWith1For4_5 = (processor4.getProductWith1() *
        processor5.getProductWith1());
    assert(productWith1For4_5 == Relation.Relation("[4]"));
```

3.5. TEST_QUANTUM_COUNTER.PY

```
# QuantumCounter/testQuantumCounter
# Nicholas Killeen,
# 19th June 2016.
# Unit tests for QuantumCounter.py.

import QuantumCounter.QuantumCounter as QuantumCounter;

def testQuantumCounter():
    # test constructor __init__, read, toString, and increment
    quantumCounter1 = QuantumCounter.QuantumCounter([0]);
    assert(quantumCounter1.read() == [0]);
    quantumCounter1.increment();
```

```
    assert(quantumCounter1.read() == [1]);
    assert(quantumCounter1.toString() == "[0]");
    quantumCounter1.increment();
    assert(quantumCounter1.read() == [0]);
    assert(quantumCounter1.toString() == "(1 - [0])");

    quantumCounter2 = QuantumCounter.QuantumCounter([-1]);
    bitState2 = quantumCounter2.read();
    assert(bitState2[0] != 0 and bitState2[0] != 1);
    quantumCounter2.increment();
    bitState2 = quantumCounter2.read();
    assert(bitState2[0] != 0 and bitState2[0] != 1);
    assert(quantumCounter2.toString() == "");

    quantumCounter3 = QuantumCounter.QuantumCounter([0, -1]);
    bitState3 = quantumCounter3.read();
    assert(bitState3[0] == 0)
    assert(bitState3[1] != 0 and bitState3[1] != 1);
    quantumCounter3.increment();
    assert(bitState3[0] == 1);
    assert(bitState3[1] != 0 and bitState3[1] != 1);

    quantumCounter4 = QuantumCounter.QuantumCounter([-1, 'v', 0.5]);
    quantumCounter4.increment();
    bitState4 = quantumCounter4.read();
    assert(bitState4[0] != 0 and bitState4[0] != 1);
    assert(bitState4[1] != 0 and bitState4[1] != 1);
    assert(bitState4[2] != 0 and bitState4[2] != 1);

    quantumCounter5 = QuantumCounter.QuantumCounter([-10, 0, 1.2]);
    assert(quantumCounter5.toString() == "(1 - [1])");
    bitState5 = quantumCounter5.read();
    assert(bitState5[0] != 0 and bitState5[0] != 1);
    assert(bitState5[1] == 0);
    assert(bitState5[2] != 0 and bitState5[2] != 1);
    quantumCounter5.increment();
    assert(quantumCounter5.toString() == "[1]");
    bitState5 = quantumCounter5.read();
    assert(bitState5[0] != 0 and bitState5[0] != 1);
    assert(bitState5[1] == 1);
    assert(bitState5[2] != 0 and bitState5[2] != 1);
    quantumCounter5.increment();
    bitState5 = quantumCounter5.read();
    assert(bitState5[0] != 0 and bitState5[0] != 1);
    assert(bitState5[1] == 0);
    assert(bitState5[2] != 0 and bitState5[2] != 1);

    quantumCounter6 = QuantumCounter.QuantumCounter([0, 0, 0, 1]);
    assert(quantumCounter6.toString() == "(1 - [0])(1 - [1])(1 - [2])[3]");
    assert(quantumCounter6.read() == [0, 0, 0, 1]);
    quantumCounter6.increment();
    assert(quantumCounter6.toString() == "(1 - [0])(1 - [1])[2](1 - [3])");
    assert(quantumCounter6.read() == [0, 0, 1, 0]);
```

```
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [0, 0, 1, 1]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [0, 1, 0, 0]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [0, 1, 0, 1]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [0, 1, 1, 0]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [0, 1, 1, 1]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [1, 0, 0, 0]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [1, 0, 0, 1]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [1, 0, 1, 0]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [1, 0, 1, 1]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [1, 1, 0, 0]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [1, 1, 0, 1]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [1, 1, 1, 0]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [1, 1, 1, 1]);
    quantumCounter6.increment();
    assert(quantumCounter6.read() == [0, 0, 0, 0]);

    quantumCounter7 = QuantumCounter.QuantumCounter([0, -1, "", 0, 0]);
    bitState7 = quantumCounter7.read();
    assert(bitState7[0] == 0);
    assert(bitState7[1] != 0 and bitState7[1] != 1);
    assert(bitState7[2] != 0 and bitState7[2] != 1);
    assert(bitState7[3] == 0);
    assert(bitState7[4] == 0);
    quantumCounter7.increment();
    bitState7 = quantumCounter7.read();
    assert(bitState7[0] == 0);
    assert(bitState7[1] != 0 and bitState7[1] != 1);
    assert(bitState7[2] != 0 and bitState7[2] != 1);
    assert(bitState7[3] == 0);
    assert(bitState7[4] == 1);
    quantumCounter7.increment();
    bitState7 = quantumCounter7.read();
    assert(bitState7[0] == 0);
    assert(bitState7[1] != 0 and bitState7[1] != 1);
    assert(bitState7[2] != 0 and bitState7[2] != 1);
    assert(bitState7[3] == 1);
    assert(bitState7[4] == 0);
    quantumCounter7.increment();
    bitState7 = quantumCounter7.read();
    assert(bitState7[0] == 0);
```

```
    assert(bitState7[1] != 0 and bitState7[1] != 1);
    assert(bitState7[2] != 0 and bitState7[2] != 1);
    assert(bitState7[3] == 1);
    assert(bitState7[4] == 1);
    quantumCounter7.increment();
    bitState7 = quantumCounter7.read();
    assert(bitState7[0] == 1);
    assert(bitState7[1] != 0 and bitState7[1] != 1);
    assert(bitState7[2] != 0 and bitState7[2] != 1);
    assert(bitState7[3] == 0);
    assert(bitState7[4] == 0);
    quantumCounter7.increment();

    quantumCounter8 = QuantumCounter.QuantumCounter([0, 1, 0, 1]);
    assert(quantumCounter8.read() == [0, 1, 0, 1]);
    quantumCounter8.increment();
    assert(quantumCounter8.read() == [0, 1, 1, 0]);
    quantumCounter8.increment();
    assert(quantumCounter8.read() == [0, 1, 1, 1]);
    quantumCounter8.increment();
    assert(quantumCounter8.read() == [1, 0, 0, 0]);

    # testing copyFrom
    quantumCounter9 = QuantumCounter.QuantumCounter([0, 5, '0', 1]);
    quantumCounter10 = QuantumCounter.QuantumCounter();
    quantumCounter10.copyFrom(quantumCounter9);
    assert(quantumCounter9.read() == quantumCounter10.read());

    # testing createCoefficientMap
    quantumCounter11 = QuantumCounter.QuantumCounter();
    quantumCounter11.createCoefficientMap([3]);
    assert(quantumCounter11.read()[3] == 0);
    quantumCounter11.increment();
    assert(quantumCounter11.read()[3] == 1);
    quantumCounter11.increment();
    assert(quantumCounter11.read()[3] == 0);

    quantumCounter12 = QuantumCounter.QuantumCounter();
    quantumCounter12.createCoefficientMap([0, 1, 2, 4]);
    bitState12 = quantumCounter12.read();
    assert(bitState12[0] == 0);
    assert(bitState12[1] == 0);
    assert(bitState12[2] == 0);
    assert(bitState12[3] != 0 and bitState12[3] != 1);
    assert(bitState12[4] == 0);

    quantumCounter13 = QuantumCounter.QuantumCounter();
    quantumCounter13.createCoefficientMap([1, 12, 2, 3]);
    quantumCounter14 = QuantumCounter.QuantumCounter(['0', 0, 0, 0, -1,
        2, 3, 5, "guacomole", "", -1.30001, 'v', 0]);

    # testing add
    quantumCounter15 = QuantumCounter.QuantumCounter(['a', 1, 2, 0, 1]);
```

```
    quantumCounter16 = QuantumCounter.QuantumCounter(['a', 1, 2, 0, 1]);
    quantumCounter15.increment();
    quantumCounter16.add(1);
    assert(quantumCounter15.read() == quantumCounter16.read());
    quantumCounter15.increment(); # 1
    quantumCounter15.increment(); # 2
    quantumCounter15.increment(); # 3
    quantumCounter15.increment(); # 4
    quantumCounter16.add(4);
    assert(quantumCounter15.read() == quantumCounter16.read());

    quantumCounter17 = QuantumCounter.QuantumCounter([0, 0, 0, 1, 'v',
        0, 1, 0, 'v']);
    quantumCounter18 = QuantumCounter.QuantumCounter([0, 0, 0, 1, 'v',
        0, 1, 0, 'v']);
    i = 0;
    while (i < 10):
        # increment 10 times
        quantumCounter17.increment();
        i += 1;
    quantumCounter18.add(10);
    assert(quantumCounter17.read() == quantumCounter17.read());
    while (i < 230):
        # increment 230 times
        quantumCounter17.increment();
        i += 1;
    quantumCounter18.add(229);
    quantumCounter18.increment();
    assert(quantumCounter17.read() == quantumCounter17.read());
```

3.6. TEST_RELATION.PY

```
# Relation/testRelation.py
# Nicholas Killeen,
# 19th June 2016.
# Unit tests for Relation.py.

import Relation.Relation as Relation;
import QuantumCounter.QuantumCounter as QuantumCounter;

def testRelation():
    # test helper function _areAllCharactersLegal
    assert(Relation._areAllCharactersLegal("") == True);
    assert(Relation._areAllCharactersLegal(" ") == True);
    assert(Relation._areAllCharactersLegal("1 0 + -") == True);
    assert(Relation._areAllCharactersLegal("012345678911 [") == True);
    assert(Relation._areAllCharactersLegal("[]1( )") == True);
    assert(Relation._areAllCharactersLegal(",") == False);
    assert(Relation._areAllCharactersLegal(". ") == False);
    assert(Relation._areAllCharactersLegal("13 16 22 4 19 a") == False);
    assert(Relation._areAllCharactersLegal("`") == False);
    assert(Relation._areAllCharactersLegal("*") == False);
    assert(Relation._areAllCharactersLegal("/ 9001") == False);
```

```
    # test helper function _areCombinationsLegal
    assert(Relation._areCombinationsLegal("") == False);
    assert(Relation._areCombinationsLegal(" + ") == False);
    assert(Relation._areCombinationsLegal(")-") == False);
    assert(Relation._areCombinationsLegal("(12) -") == False);
    assert(Relation._areCombinationsLegal("(144)-     ") == False);
    assert(Relation._areCombinationsLegal("(1) (9)") == False);
    assert(Relation._areCombinationsLegal("(4) 2") == False);
    assert(Relation._areCombinationsLegal("2 [1]") == False);
    assert(Relation._areCombinationsLegal("(1]+ 3") == False);
    assert(Relation._areCombinationsLegal("2 [1]") == False);
    assert(Relation._areCombinationsLegal("[") == False);
    assert(Relation._areCombinationsLegal("]") == False);
    assert(Relation._areCombinationsLegal("[1)") == False);
    assert(Relation._areCombinationsLegal("(1 2)") == False);
    assert(Relation._areCombinationsLegal("[3 1]") == False);
    assert(Relation._areCombinationsLegal("(     )(12)") == False);
    assert(Relation._areCombinationsLegal("()") == False);
    assert(Relation._areCombinationsLegal("[12]") == True);
    assert(Relation._areCombinationsLegal("[123456789](0)") == True);
    assert(Relation._areCombinationsLegal("1 + 1") == True);
    assert(Relation._areCombinationsLegal("2+n") == True);
    assert(Relation._areCombinationsLegal("n+(5)") == True);
    assert(Relation._areCombinationsLegal("[5]+(((n)))") == True);
    assert(Relation._areCombinationsLegal("((1+((1)(( n)))))") == True);

    # test helper function _areNumbersLegal
    assert(Relation._areNumbersLegal("") == True);
    assert(Relation._areNumbersLegal("tortilla") == True);
    assert(Relation._areNumbersLegal("0 1-- 2, three 1007") == True);
    assert(Relation._areNumbersLegal("007") == False);
    assert(Relation._areNumbersLegal("0*0*1") == True);
    assert(Relation._areNumbersLegal("00     12") == False);
    assert(Relation._areNumbersLegal("100000 600001") == True);

    # test helper function _isNestingLegal
    assert(Relation._isNestingLegal("") == True);
    assert(Relation._isNestingLegal("()") == True);
    assert(Relation._isNestingLegal("      ") == True);
    assert(Relation._isNestingLegal(" ( ( )  ) ") == True);
    assert(Relation._isNestingLegal("(a(x)+) (]") == True);
    assert(Relation._isNestingLegal("([[1()]()[2]])") == True);
    assert(Relation._isNestingLegal("(()()") == False);
    assert(Relation._isNestingLegal(")( )( ") == False);
    assert(Relation._isNestingLegal(")(())(") == False);
    assert(Relation._isNestingLegal("(a)v)") == False);

    # test helper function _recursivelyReplace
    assert(Relation._recursivelyReplace("ab", "a", "b") == "bb");
    assert(Relation._recursivelyReplace("ab", "ab", "ab") == "ab");
    assert(Relation._recursivelyReplace("23232323", "2323", "23") ==
        "23");
```

```
    assert(Relation._recursivelyReplace("110111101 10011111111", "11",
        "1") == "10101 1001");
    assert(Relation._recursivelyReplace("", "a", "cd") == "");
    assert(Relation._recursivelyReplace("", "", "1") == "");
    assert(Relation._recursivelyReplace("", "n", "") == "");
    assert(Relation._recursivelyReplace("x", "", "") == "x");

    # test test-helper function, _areArraysEqual
    assert(_areArraysEqual([0], [1]) == False);
    assert(_areArraysEqual([1, 2], [1]) == False);
    assert(_areArraysEqual([1, 2, 3, 4], [1, 3, 2, 4]) == True);
    assert(_areArraysEqual([0, 0], [1]) == False);
    assert(_areArraysEqual([0, 199], [199, 0]) == True);
    assert(_areArraysEqual([], []) == True);
    assert(_areArraysEqual([12], []) == False);
    assert(_areArraysEqual([12, 13, 13], [13, 12, 12]) == False);
    assert(_areArraysEqual([12, 13, 13], [13, 12, 12]) == False);
    assert(_areArraysEqual([1, 2, 3, 55, 102, 1042, 4, 16], [1, 2, 3,
        55, 1042, 102, 4, 16]) == True);
    assert(_areArraysEqual([1, 2, 3, 55, 102, 1043, 4, 16], [1, 2, 3,
        55, 1042, 102, 4, 16]) == False);

    # test getSymbolList
    relation1 = Relation.Relation("[1] + [0](1 - [1])");
    symbolsInRelation1 = relation1.getSymbolList();
    assert(_areArraysEqual(symbolsInRelation1, [0, 1]));

    relation2 = Relation.Relation("1");
    symbolsInRelation2 = relation2.getSymbolList();
    assert(_areArraysEqual(symbolsInRelation2, []));

    relation3 = Relation.Relation("0 + [1]");
    symbolsInRelation3 = relation3.getSymbolList();
    assert(_areArraysEqual(symbolsInRelation3, [1]));

    relation4 = Relation.Relation("0 + [5](1 - [2])(1 - [4][7])");
    symbolsInRelation4 = relation4.getSymbolList();
    assert(_areArraysEqual(symbolsInRelation4, [2, 4, 5, 7]));

    relation5 = Relation.Relation("(1 - [3])([12] + [15](1 - [12]))");
    symbolsInRelation5 = relation5.getSymbolList();
    assert(_areArraysEqual(symbolsInRelation5, [3, 12, 15]));

    relation6 = Relation.Relation("[1][2][100] + [9001][1](1 - [2])(1 -"
        " [100])");
    symbolsInRelation6 = relation6.getSymbolList();
    assert(_areArraysEqual(symbolsInRelation6, [1, 2, 100, 9001]));

    # test evaluate
    relation7 = Relation.Relation("1");
    assert(relation7.evaluate() == 1);

    relation8 = Relation.Relation("0");
```

```
    assert(relation8.evaluate() == 0);

    relation9 = Relation.Relation("(0)(1)");
    assert(relation9.evaluate() == 0);

    relation10 = Relation.Relation("(0)(1) + 1 + (0)(1 - (1))");
    assert(relation10.evaluate() == 1);

    relation11 = Relation.Relation("(1)(1)(0) + (1)(1 - (1))(1 - ((0)(1"
        "))) + (1 - (1))(1 - (1))");
    assert(relation11.evaluate() == 0);

    relation12 = Relation.Relation("(0)(1 - (0)) + (0)(1 - (1)) + (1)("
        "1 - (0))(1 - (1 - (1)))");
    assert(relation12.evaluate() == 1);

    # test insertArguments, "==" operator __eq__, copyFrom, and the "!="
    # comparator __ne__
    quantumCounter13 = QuantumCounter.QuantumCounter([1, 0]);
    relation13 = Relation.Relation("[1]");
    relation13.insertArguments(quantumCounter13);
    assert(relation13.evaluate() == 0);

    quantumCounter14 = QuantumCounter.QuantumCounter([0, 0]);
    relation14 = Relation.Relation("[0] + [0](1 - [1])");
    relation14.insertArguments(quantumCounter14);
    assert(relation14.evaluate() == 0);

    quantumCounter15 = QuantumCounter.QuantumCounter([1]);
    relation15 = Relation.Relation("[0](1 - [1]) + [1](1 - [0])");
    relation15.insertArguments(quantumCounter15);
    relation16 = Relation.Relation("(1 - [1])");
    assert((relation15 == relation16) == True);
    assert((relation15 != relation16) == False);
    relation16Copy = Relation.Relation();
    relation16Copy.copyFrom(relation16);
    assert(relation16 == relation16Copy);

    relation17 = Relation.Relation("[4]");
    relation18 = Relation.Relation("0");
    assert((relation17 == relation18) == False);
    assert((relation17 != relation18) == True);
    quantumCounter17 = QuantumCounter.QuantumCounter([1, 0, -1, 0, 0]);
    relation17.insertArguments(quantumCounter17);
    assert((relation17 == relation18) == True);
    assert((relation17 != relation18) == False);
    assert((relation17 == relation17) == True);
    assert((relation17 != relation17) == False);
    assert((relation18 == relation18) == True);
    assert((relation18 != relation18) == False);
    relation17Copy = Relation.Relation();
    relation17Copy.copyFrom(relation17);
    assert(relation17 == relation17Copy);
```

```
    relation19 = Relation.Relation("1");
    relation20 = Relation.Relation("[2][1][111] + (1 - [2])[111](1 - [1"
        "]) + (0)(1 - [6]) + [1][2](1 - [111]) + (1 - [1])(1 - [111]) +"
        " [1][111](1 - [2]) + [2][111](1 - [1]) + 0 + [1](1 - [111])(1 "
        "- [2])");
    relation21 = Relation.Relation("[2][1][111] + (1 - [2])[111](1 - [1"
        "]) + (0)(1 - [6]) + [1][2](1 - [111]) + (1 - [1])(1 - [111]) +"
        " [1][111](1 - [2]) + [2][111](1 - [1]) + 0 + [1](1 - [111])(1 "
        "- [1])");
    assert((relation19 == relation20) == True);
    assert((relation19 != relation20) == False);
    assert((relation20 == relation21) == False);
    assert((relation20 != relation21) == True);

    relation22 = Relation.Relation("[4][0][1] + (1 - [0])[1](1 - [4])");
    quantumCounter22 = QuantumCounter.QuantumCounter(['v', 'v', 'v',
        'v', 1, 'v', 1]);
    relation23 = Relation.Relation("(1)[0][1] + (1 - [0])[1](1 - (1))");
    assert((relation22 == relation23) == False);
    relation22.insertArguments(quantumCounter22);
    assert(relation22 == relation23);
    relation24 = Relation.Relation();
    relation24.copyFrom(relation22);
    assert(relation23 == relation24);

    relation25 = Relation.Relation("0 + [1](1 - [2])(1 - [0][3])");
    relation26 = Relation.Relation();
    relation26.copyFrom(relation25);
    relation27 = Relation.Relation();
    relation27.copyFrom(relation26);
    assert(relation25 == relation26);
    assert(relation25 == relation27);
    assert((relation26 != relation27) == False);
    quantumCounter25 = QuantumCounter.QuantumCounter([]);
    relation25.insertArguments(quantumCounter25);
    assert(relation25 == relation26);
    quantumCounter26 = QuantumCounter.QuantumCounter(["1"]);
    relation26.insertArguments(quantumCounter26);
    assert(relation25 == relation26);
    assert(relation26 == relation27);
    quantumCounter27 = QuantumCounter.QuantumCounter([0, 1, 0, 1]);
    relation27.insertArguments(quantumCounter27);
    assert(relation27.evaluate() == 1);

    # test the "*" operator __mul__
    relation28 = Relation.Relation("[0]");
    relation29 = Relation.Relation("[0]");
    relation30 = relation28 * relation29;
    assert(relation28 * relation29 == relation28);
    assert(relation28 * relation29 == relation29);
    relation31 = Relation.Relation("[1]");
    relation32 = relation31 * relation30;
```

```
    relation33 = relation30 * relation31;
    relation34 = Relation.Relation("[0][1]");
    assert(relation32 == relation33);
    assert(relation34 == relation33);
    assert(relation34 == relation32);
    assert(relation34 != relation28);

    relation35 = Relation.Relation("[0] + [1](1 - [0])");
    relation36 = Relation.Relation("0");
    relation37 = Relation.Relation("1");
    relation38 = Relation.Relation("(0) + ((([1])))");
    relation39 = Relation.Relation("([1])");
    assert(relation35 * relation36 == relation36);
    assert(relation35 != relation36);
    assert(relation35 * relation35 == relation35);
    assert(relation35 * relation37 == relation35);
    assert(relation37 * relation35 == relation35);
    assert(relation36 * relation37 == relation36);
    assert(relation36 * relation38 == relation36);
    assert(relation37 * relation38 == relation38);
    assert(relation38 * relation38 == relation38);
    assert(relation35 * relation38 == relation39);
    assert(relation38 * relation35 == relation39);
    assert(relation38 * relation39 == relation39);

    # test eliminateSymbol
    relation39 = Relation.Relation("[0]");
    relation39.eliminateSymbol(0);
    assert(relation39.evaluate() == 0);

    relation40 = Relation.Relation("[0](1 - [1]) + (1 - [0])(1 - [1])");
    relation40.eliminateSymbol(1);
    relation41 = Relation.Relation("0");
    assert(relation40 == relation41);

    relation42 = Relation.Relation("[9](1 - [1]) + [9][1](1 - [2][3])");
    relation42.eliminateSymbol(1);
    relation43 = Relation.Relation("[9](1 - [2][3])");
    assert(relation42 == relation43);

    relation44 = Relation.Relation("0");
    relation44.eliminateSymbol(5);
    assert(relation44.evaluate() == 0);

    relation45 = Relation.Relation("[10][2][3] + [10](1 - [2])(1 - [3])"
        " + (1 - [10])(1 - [2])");
    relation46 = Relation.Relation("(1 - [2])(1 - [3])");
    relation45.eliminateSymbol(10);
    assert(relation45 == relation46);

    # test the binary "+" operator __add__
    relation46 = Relation.Relation("0");
    relation47 = Relation.Relation("1");
```

```
    relation48 = Relation.Relation("[2]");
    relation49 = Relation.Relation("(1 - [2])");
    assert(relation46 + relation46 == relation46);
    assert(relation46 + relation47 == relation47);
    assert(relation48 + relation49 + relation46 == relation47);

    relation50 = Relation.Relation("[12][11]");
    relation51 = Relation.Relation("(1 - [12])[11][4]");
    relation52 = Relation.Relation("(1 - [12])[11](1 - [4])");
    relation53 = Relation.Relation("(1 - [12])[11]");
    relation54 = Relation.Relation("[11]");
    assert(relation50 + relation51 + relation52 == relation54);
    assert(relation52 + relation50 + relation51 == relation54);
    assert(relation53 == relation51 + relation52);
    assert(relation54 == relation50 + relation53);

    relation55 = Relation.Relation("[102](1 - [0])");
    relation56 = Relation.Relation("[0](1 - [1][102])");
    relation57 = Relation.Relation("[102](1 - [0])(1 - [1]) + [102](1 -"
        " [0])[1] + [0](1 - [1][102])");
    relation58 = Relation.Relation("[102](1 - [0])(1 - [4]) + [102](1 -"
        " [0])[1] + [0](1 - [1][102])");
    assert(relation55 + relation56 == relation57);
    assert(relation55 + relation56 != relation58);

    # test binary "-" operator __sub__
    relation57 = Relation.Relation("[102]");
    relation58 = Relation.Relation("(1 - [102])");
    relation59 = Relation.Relation("0");
    relation60 = Relation.Relation("1");
    assert(relation57 - relation57 == relation59);
    assert(relation60 - relation57 == relation58);
    assert(relation60 - relation58 == relation57);
    assert(relation60 - (relation60 - relation58) == relation58);
    assert(relation60 - (relation60 - relation57) == relation57);

    relation61 = Relation.Relation("[3](1 - [4]) + (1 - [3])[4]");
    relation62 = Relation.Relation("[12]([3][4] + (1 - [3])(1 - [4])) +"
        " (1 - [12])([3][4] + (1 - [3])(1 - [4]))");
    relation63 = Relation.Relation("1");
    assert(relation63 - relation61 == relation62);
    assert(relation63 - relation62 == relation61);

    relation64 = Relation.Relation("[0][1](1 - [2])[4] + (1 - [4])");
    relation65 = Relation.Relation("(1 - [4])");
    relation66 = Relation.Relation("[0][1](1 - [2])[4]");
    assert(relation64 - relation65 == relation66);
    assert(relation64 - relation66 == relation65);

    # test isValid, optimise
    relation67 = Relation.Relation("0");
    relation67Copy = Relation.Relation();
    relation67Copy.copyFrom(relation67);
```

```
    relation67Copy.optimise();
    assert(relation67.isValid() == True);
    assert(relation67 == relation67Copy);

    relation68 = Relation.Relation("1");
    relation68Copy = Relation.Relation();
    relation68Copy.copyFrom(relation68);
    relation68Copy.optimise();
    assert(relation68.isValid() == True);
    assert(relation68 == relation68Copy);

    relation69 = Relation.Relation("(1) + 0");
    relation69Copy = Relation.Relation();
    relation69Copy.copyFrom(relation69);
    relation69Copy.optimise();
    assert(relation69.isValid() == True);
    assert(relation69 == relation69Copy);

    relation70 = Relation.Relation("(1)(1)");
    relation70Copy = Relation.Relation();
    relation70Copy.copyFrom(relation70);
    relation70Copy.optimise();
    assert(relation70.isValid() == True);
    assert(relation70 == relation70Copy);

    relation71 = Relation.Relation("(((1)))(0)");
    relation71Copy = Relation.Relation();
    relation71Copy.copyFrom(relation71);
    relation71Copy.optimise();
    assert(relation71.isValid() == True);
    assert(relation71 == relation71Copy);

    relation72 = Relation.Relation("(((1)))(0) - (0)");
    relation72Copy = Relation.Relation();
    relation72Copy.copyFrom(relation72);
    relation72Copy.optimise();
    assert(relation72.isValid() == True);
    assert(relation72 == relation72Copy);

    relation73 = Relation.Relation("[0]");
    relation73Copy = Relation.Relation();
    relation73Copy.copyFrom(relation73);
    relation73Copy.optimise();
    assert(relation73.isValid() == True);
    assert(relation73 == relation73Copy);

    relation74 = Relation.Relation("[10]");
    relation74Copy = Relation.Relation();
    relation74Copy.copyFrom(relation74);
    relation74Copy.optimise();
    assert(relation74.isValid() == True);
    assert(relation74 == relation74Copy);
```

```
    relation75 = Relation.Relation("[104]");
    relation75Copy = Relation.Relation();
    relation75Copy.copyFrom(relation75);
    relation75Copy.optimise();
    assert(relation75.isValid() == True);
    assert(relation75 == relation75Copy);

    relation76 = Relation.Relation("1 - [99]");
    relation76Copy = Relation.Relation();
    relation76Copy.copyFrom(relation76);
    relation76Copy.optimise();
    assert(relation76.isValid() == True);
    assert(relation76 == relation76Copy);

    relation77 = Relation.Relation("1 - [13][12]");
    relation77Copy = Relation.Relation();
    relation77Copy.copyFrom(relation77);
    relation77Copy.optimise();
    assert(relation77.isValid() == True);
    assert(relation77 == relation77Copy);

    relation78 = Relation.Relation("(1 - [13][1])[13]");
    relation78Copy = Relation.Relation();
    relation78Copy.copyFrom(relation78);
    relation78Copy.optimise();
    assert(relation78.isValid() == True);
    assert(relation78 == relation78Copy);

    relation79 = Relation.Relation("[0] + (1 - [0])[1]");
    relation79Copy = Relation.Relation();
    relation79Copy.copyFrom(relation79);
    relation79Copy.optimise();
    assert(relation79.isValid() == True);
    assert(relation79 == relation79Copy);

    relation80 = Relation.Relation("[0](1 - [2]) + (1 - [0])[1](1 - [2]"
        " + [2])");
    relation80Copy = Relation.Relation();
    relation80Copy.copyFrom(relation80);
    relation80Copy.optimise();
    assert(relation80.isValid() == True);
    assert(relation80 == relation80Copy);

    relation81 = Relation.Relation("     0 ");
    relation81Copy = Relation.Relation();
    relation81Copy.copyFrom(relation81);
    relation81Copy.optimise();
    assert(relation81.isValid() == True);
    assert(relation81 == relation81Copy);

    relation82 = Relation.Relation(" (          1 )+  0");
    relation82Copy = Relation.Relation();
    relation82Copy.copyFrom(relation82);
```

```
    relation82Copy.optimise();
    assert(relation82.isValid() == True);
    assert(relation82 == relation82Copy);

    relation83 = Relation.Relation("1    +   (0)(1)");
    relation83Copy = Relation.Relation();
    relation83Copy.copyFrom(relation83);
    relation83Copy.optimise();
    assert(relation83.isValid() == True);
    assert(relation83 == relation83Copy);

    relation84 = Relation.Relation("                                    "
        "                                                               "
        "                            [0]                                "
        "                                                               "
        "                                                               "
        "            +                 (1 - [0])                        "
        "                                                             ");
    relation84Copy = Relation.Relation();
    relation84Copy.copyFrom(relation84);
    relation84Copy.optimise();
    assert(relation84.isValid() == True);
    assert(relation84 == relation84Copy);

    relation85 = Relation.Relation("((                ([0])))([0])+(1-"
        "[0])");
    relation85Copy = Relation.Relation();
    relation85Copy.copyFrom(relation85);
    relation85Copy.optimise();
    assert(relation85.isValid() == True);
    assert(relation85 == relation85Copy);

    relation86 = Relation.Relation("[0][2] -  0 - ([2][0][1] - [1])");
    relation86Copy = Relation.Relation();
    relation86Copy.copyFrom(relation86);
    relation86Copy.optimise();
    assert(relation86.isValid() == True);
    assert(relation86 == relation86Copy);

    relation87 = Relation.Relation("-1");
    assert(relation87.isValid() == False);

    relation88 = Relation.Relation("-2");
    assert(relation88.isValid() == False);

    relation89 = Relation.Relation("2");
    assert(relation89.isValid() == False);

    relation90 = Relation.Relation("[0] + [1]");
    assert(relation90.isValid() == False);

    relation91 = Relation.Relation("[0] + [0]");
    assert(relation91.isValid() == False);
```

```
    relation92 = Relation.Relation("");
    assert(relation92.isValid() == False);

    relation93 = Relation.Relation("0.0");
    assert(relation93.isValid() == False);

    relation94 = Relation.Relation("all men are mortals");
    assert(relation94.isValid() == False);

    relation95 = Relation.Relation("((1)");
    assert(relation95.isValid() == False);

    relation96 = Relation.Relation("2(1)");
    assert(relation96.isValid() == False);

    relation97 = Relation.Relation("[ 0]");
    assert(relation97.isValid() == False);

    relation98 = Relation.Relation("[10 2]");
    assert(relation98.isValid() == False);

    relation99 = Relation.Relation("[2 ]");
    assert(relation99.isValid() == False);

    relation100 = Relation.Relation("[2+1]");
    assert(relation100.isValid() == False);

    relation101 = Relation.Relation("[12] + ()(1 - [12])");
    assert(relation101.isValid() == False);

    relation102 = Relation.Relation("[12] + (1(1) 1)");
    assert(relation102.isValid() == False);

    relation103 = Relation.Relation("])");
    assert(relation103.isValid() == False);

    relation104 = Relation.Relation("[-1])");
    assert(relation104.isValid() == False);

    relation105 = Relation.Relation("[00]");
    assert(relation105.isValid() == False);

    relation106 = Relation.Relation("00");
    assert(relation106.isValid() == False);

    relation107 = Relation.Relation("[01]");
    assert(relation107.isValid() == False);

    relation108 = Relation.Relation("+");
    assert(relation108.isValid() == False);

    relation109 = Relation.Relation("-");
```

```
    assert(relation109.isValid() == False);

    relation110 = Relation.Relation("[12]a");
    assert(relation110.isValid() == False);

    relation111 = Relation.Relation("[12] + b");
    assert(relation111.isValid() == False);

    relation112 = Relation.Relation("[12] + ");
    assert(relation112.isValid() == False);

    relation113 = Relation.Relation("1 +*");
    assert(relation113.isValid() == False);

    relation114 = Relation.Relation("(1)*(1)");
    assert(relation114.isValid() == False);

    relation115 = Relation.Relation("[");
    assert(relation115.isValid() == False);

    relation116 = Relation.Relation(")");
    assert(relation116.isValid() == False);

    relation117 = Relation.Relation("([1)]");
    assert(relation117.isValid() == False);

    relation118 = Relation.Relation("1 - [13][3]     )");
    assert(relation118.isValid() == False);

    relation119 = Relation.Relation("(1) (1)");
    assert(relation119.isValid() == False);

    relation120 = Relation.Relation("9 / 9");
    assert(relation120.isValid() == False);

    relation121 = Relation.Relation("1/1");
    assert(relation121.isValid() == False);

    relation122 = Relation.Relation("[1] + (1 - [1.])");
    assert(relation122.isValid() == False);

    relation123 = Relation.Relation("[1] + l (1 - [1])");
    assert(relation123.isValid() == False);

    relation124 = Relation.Relation("{1 + 0}");
    assert(relation124.isValid() == False);

    relation125 = Relation.Relation("shredded lettuce");
    assert(relation125.isValid() == False);

    relation126 = Relation.Relation("\"\"");
    assert(relation126.isValid() == False);
```

```
    relation127 = Relation.Relation("assert(False)");
    assert(relation127.isValid() == False);

    relation128 = Relation.Relation("eval(assert(False))");
    assert(relation128.isValid() == False);

    relation129 = Relation.Relation("1 + 0^1");
    assert(relation129.isValid() == False);

    relation130 = Relation.Relation("1 -- 0");
    assert(relation130.isValid() == False);

    relation131 = Relation.Relation("0 +- 0");
    assert(relation131.isValid() == False);

    relation132 = Relation.Relation("0 +- 1");
    assert(relation132.isValid() == False);

    relation133 = Relation.Relation("(((((((1))))))) )");
    assert(relation133.isValid() == False);

    relation134 = Relation.Relation("0\t");
    assert(relation134.isValid() == False);

    relation135 = Relation.Relation("0 0");
    assert(relation135.isValid() == False);

    relation136 = Relation.Relation("0 0");
    assert(relation136.isValid() == False);

    relation137 = Relation.Relation("1[0]");
    assert(relation137.isValid() == False);

    relation138 = Relation.Relation("-[0]");
    assert(relation138.isValid() == False);

    relation139 = Relation.Relation("[0][1] + [0](1 - [1]) + (1-[0])[1] "
        "+ (1-[0])(1 - [1]) + [2](1 - [1])");
    assert(relation139.isValid() == False);

    relation140 = Relation.Relation("((([0]) + (1 - [0])) + 1(0))");
    assert(relation140.isValid() == False);

    relation141 = Relation.Relation(")1(");
    assert(relation141.isValid() == False);

    relation142 = Relation.Relation("]4[");
    assert(relation142.isValid() == False);

    relation143 = Relation.Relation("[6] -(");
    assert(relation143.isValid() == False);

    relation144 = Relation.Relation("1(1 - [1])");
```

```
    assert(relation144.isValid() == False);

    relation145 = Relation.Relation("(1 - [1])1");
    assert(relation145.isValid() == False);

# Helper function to test two unsorted arrays for equality.
def _areArraysEqual(arrayA, arrayB):
    hasEncounteredDiscrepancy = False;

    # create a list of all symbols in A
    symbolListConcatenation = "";
    index = 0;
    lengthOfA = len(arrayA);
    while (index < lengthOfA):
        currentElement = str(arrayA[index]);
        assert(currentElement.find('[') == -1);
        assert(currentElement.find(']') == -1);
        item = '[' + currentElement + ']';
        if (symbolListConcatenation.find(item) != -1):
            hasEncounteredDiscrepancy = True;
        symbolListConcatenation += item;
        index += 1;

    # eliminate every symbol in A that is also present in B
    index = 0;
    lengthOfB = len(arrayB);
    while (index < lengthOfB):
        currentElement = str(arrayB[index]);
        assert(currentElement.find('[') == -1);
        assert(currentElement.find(']') == -1);
        item = '[' + currentElement + ']';
        if (symbolListConcatenation.find(item) == -1):
            hasEncounteredDiscrepancy = True;
        symbolListConcatenation = symbolListConcatenation.replace(
            item, "");
        index += 1;

    if (symbolListConcatenation != ""):
        hasEncounteredDiscrepancy = True;

    return not hasEncounteredDiscrepancy;
```

4. HIGH LEVEL TESTING

4.1. ON WHAT HAS BEEN DONE

Hereto has occurred deskchecking, black-box testing of data structures (and development of accompanying unit tests), and ongoing debugging and prodding of the system. What remains, optimally, is alpha and beta testing. Indeed, in the perfect scenario, the PLE would be distributed to a select group of end users to play around with -- but Python distribution is simply too much of a hassle, and it is wasteful to attempt compilation for distribution of anything less than an already tested program. If ran fully, the course of this development would unfavourably see a beta release published as the final product, and iterated upon through ongoing maintenance. Irrespectively, in the scenario at hand, taking this report to be chronological (as it tries to be), it remains that live tests and acceptance tests are to be performed. Live tests will be found explicitly under this section, and acceptance tests tacitly, although the latter will be discussed in `V. 6. with reference to the success or failure of the live tests.

4.2. LIVE TESTS PLAN

From a list of features, individual test routines are designed to test the darkest corners of the system. Anything pertinent to the quality of the software will be reported on, including things such as response times. A list of test areas that tests must be designed to cover follows:

- Adding, removing, and clearing facts and symbols.

- Saving / Loading.

- Checking validity of facts.

- Parsing values of symbol numbers.

- Standard functionality.

- Quantum functionality.

- Invalid data, and mixes of invalid data.

- Complex and great amounts of data.

- Differing window sizes.

When, inevitably, errors are exposed, they must be corrected (if they are to be) in such a way that they cannot conceivably interfere with the program in unexpected ways. Particular attention must thus be observed when modifying the source. Odd behaviour highlighted.

4.3. LIVE TESTS (ACTIONS)

Check 1. Adding and removing individual facts works.
Check 2. Values of facts are retained through deletion and addition.
Check 3. Descriptions are saved to the selected fact, and deleting and adding facts interacts correctly with descriptions.
Check 4. Resizing the window resizes the contents correctly.
Check 5. Adding and removing individual symbols works.
Check 6. The selection of symbols has no bearing on the selected relation, and vice versa.
Check 7. Values of symbol descriptions are retained through deletion and addition of symbols.
Check 8. Symbols and relations can be individually cleared.
Check 9. Pressing the "Clear All" or "Remove" buttons for either facts or symbols, when the respective fields are empty, results in no change.
Check 10. Typing non-numerical values as symbol numbers results in no change.
Check 11. Numerical values are accepted as symbol numbers, and strings are correctly truncated once they exceed three characters in length. After any addition to a length 3 string, the caret becomes focussed at the beginning of the line, instead of retaining its old position.
Check 12. Dragging text into the symbol number is prevented, **although the cursor is such that the event appears not prevented.**
Check 13. Values of symbol numbers are retained through deletion and addition of symbols.
Check 14. The "Toggle Filter" button correctly toggles the selected symbols colouring.
Check 15. Pasting the numerical value "12345" into a symbol-number field causes it to take the value of "123", quite intuitively. Results are correctly correspondent for values "a1b2c3d4", "0a1b2c3d4". Pasting with the caret in other positions in symbol-numbers with already existing values works similarly.
Check 16. Pasting symbol-number values with new-lines, such as "0\n1" cause a permanent (until more than 3 digits are entered) empty-text node to be prepended within the value, causing significant alignment errors.
Check 17. Facts are correctly identified as being valid, invalid, or too long to validate. The response time appears perfectly acceptable with the current limit to the number of symbols for validation (7). The following facts are correctly validated (and accordingly coloured), but appended afterwards are failures:
VALID
- "0=0"
- "[99] = 0 + 0"
- "1 =1"
- "0 = 1" (notice that although this is objectively incorrect, it is still a valid assertion on behalf of the user)
- "[0] =1"
- "[0]=[1][2](1 - [3])"

```
          - "[0](1-[1]) + [1](1 - [0])[2] = [4][3]"
          - "0=[1][2][3][4][5][100][7]"
          - "0=[1][2][3][4][5][100][7][100]"
          INVALID
          - "+1=+1"
          - "+[0]=0"
          - "a=b"
          - "1"
          - "b"
          - "1 =  1" (this result was not initially expected, which should
                      be noted, although it does not render this
                      functionality incorrect or even unintentional)
          - "[0] = [ 1]"
          - "[0] + [0] = 1"
          - "[0] + [1] = 1"
          - "[0] + [1] = [0]"
          - "[0] = 1 + 1"
          - " 0-1 = -[0]"
          - "(0-1)=(0-[0])"
          - "[1]=[0"
          - ""
          - "\n"
          UNCHECKED
          - "0=[1][2][3][4][5][100][7][8]"
          - "bacon=[1][2][3][4][5][100][7][8]"
          INCORRECT CLASSIFICATION
          - "[1]=[0" and "0]=1" raise the UNCHECKED flag (orange), when
          they are easily recognisable as incorrect. Silently, both cases
          throw an error in the Python command window (only made visible
          for debugging), raising "IndexError: list index out of range" in
          getSymbolList, line 93, reading:
          "currentSymbolString = currentSymbolString.split('[')[1];"
          - "0=[10000000]" is classified as VALID, with significant
          responsiveness issues (hanging for approximately fifteen
          seconds). The classification is technically correct, but, the
          symbol dictionary only supports numbers from 0 to 999, so the
          fact should be classed as invalid. Further, responsiveness is
          linearly proportional to the value contained with the brackets,
          where this particular case takes approximately 20 seconds.
Check 18. Using the same data as from Check 17, the list of symbols
          involved in a selected fact is correctly alphabetised
          (numerically), and displays, in general, the symbols correctly,
          according to whether the relation is VALID or belongs to one of
          the other two categories.
Check 19. Multiple definitions of symbols in the dictionary cause each of
          definitions to be listed as being involved in a relation --
          retaining their order.
Check 20. An empty request raises an alert "The request is malformed.", a
          behaviour mimicked also by the following invalid requests:
          - "a"
          - " "
          - "1=1"
          - "[0"
```

```
          - "0]"
          - "[0][1][a]"
          - "[ 0]"
Check 21. Saving, closing, and loading mixtures of valid, invalid, and
          unchecked relations, as well as numerous symbols with varying
          toggle states; functions correctly. Particularly, the following
          files can be saved and loaded successfully:
          - file1
            RELATIONS: None
            SYMBOLS: None
          - file2
            RELATIONS: "[0][1] = 0", "Red dogs don't exist.";
                       "[0][1][2][3][4][5][6][7][8][9][10] = 1", "desc2"
                       "a = 1", "intentionally invalid"
                       "", ""
            SYMBOLS: [0], "red", determinate;
                     [1], "dogs", quantum;
                     [], "", determinate;
Check 22. Saving to an invalid file path ("", "\d/") correctly alerts the
          user "The file path is invalid."
Check 23. Loading from an invalid or non-existent file path correctly
          alerts the user "could not find file" ("", "a.txt", "\d/").
Check 24. Empty files, and incorrectly encoded files ("",
          "aW5jb3JyZWN0bHkgZW5jb2RlZA==", "'a'") are identified as being
          malformed, and the user is alerted.
```

A table of what is believed to be all incorrect behaviour (excluding the higher order functionality, which will be soon tested) can now be created.

Incorrect Behaviour		
Revealing check	Description	Priority
11	On writing any text to a symbol-number input field which is already of length 3 causes the caret to jump to the beginning of the string, perhaps interfering with typing.	Extremely low.
12	A "copy" cursor is displayed rather than a "no-drop" when dragging text into a symbol-number input field.	Extremely low.
16	In symbol-number fields, pasting strings that contain new-lines cause empty text-nodes to be created within the contenteditable div, causing the symbol-number field to exceed its standard height of 1em, and overflow adjacent elements. Likewise when all characters of the field are dragged from the node.	Medium.
17	Certain relations that ought to be easily recognisable as INVALID (particularly, those with mismatched square brackets), are coloured as UNCHECKED by the input validator, and raise non-terminal errors in Python.	Medium.

17	High symbol values, when inputted into relation fields, cause significant responsiveness issues when their validity is automatically checked.	Medium.
19	Redefinitions of symbols in the dictionary are not disallowed (although they behave in a way that would be correct were this intentional).	Extremely low.

4.4. LIVE TESTS (DATA)

Looking to the wonderfully nonsensical Lewis Carroll, one finds a great deal of simple exercises in the domain of propositional logic. Let some few be selected, and have their results verified as live-data, thus proving the success of the program at a system-level (since such a process of verification involves the brain-teasing interpretation of facts):

```
Quaesitum: All the animals in the yard gnaw bones.
(1) I trust every animal that belongs to me;
(2) Dogs gnaw bones;
(3) I admit no animals into my study, unless they will beg when told to do
so;
(4) All the animals in the yard are mine;
(5) I admit every animal, that I trust, into my study;
(6) The only animals, that are really willing to beg when told to do so,
are dogs.

Quaesitum: No badger can guess a conundrum.
(1) Animals are always mortally offended if I fail of notice them;
(2) The only animals that belong to me are in that field;
(3) No animal can guess a conundrum, unless it has been properly trained in
a Board-School;
(4) None of the animals in that field are badgers;
(5) When an animal is mortally offended, it always rushes about wildly and
howls;
(6) I never notice any animal, unless it belongs to me;
(7) No animal, that has been properly trained in a Board-School, ever
rushes about wildly and howls.

Quaesitum: I always avoid a kangaroo.
(1) The only animals in this house are cats;
(2) Every animal is suitable for a pet, that loves to gaze at the moon;
(3) When I detest an animal, I avoid it;
(4) No animals are carnivorous, unless they prowl at night;
(5) No cat fails to kill mice;
(6) No animals ever take to me, except what are in this house;
(7) Kangaroos are not suitable for pets;
(8) None but carnivora kill mice;
(9) I detest animals that do not take to me;
(10) Animals, that prowl at night, always love to gaze at the moon.
```

Attached digitally are saves for each of these scenarios, as translated in the PLE.

The program must also be tested against specific odd combinations of input, so let a full list of cases to be tested appear:

```
Check 1.  The correct conclusions are reached in the three scenarios (prior
          page): "All the animals in the yard gnaw bones", "No badger can
          guess a conundrum", and "I always avoid a kangaroo"
Check 2.  The following scenarios meet the expected output:
          - FACTS:    ""
            REQUEST:  "0"
            RESPONSE: "0"
          - FACTS:    ""
            REQUEST:  "1"
            RESPONSE: "1"
          - FACTS:    "0 = 1"
            REQUEST:  "0"
            RESPONSE: "0 = 0", "0 = 1"
          - FACTS:    "0 = 1"
            REQUEST:  "1"
            RESPONSE: "1 = 0", "0 = 1"
          - FACTS:    ""
            REQUEST:  "[0]"
            RESPONSE: "0/0"
          - FACTS:    "[0] = 0"
            REQUEST:  "[1]"
            RESPONSE: "[1] = 0/0(1 - [0])", "0 = [0]"
```

4.5. USER FEEDBACK

On the acquisition of user feedback, it can be said that, of the five pillars of the system -- hardware, software, procedures, data, and hardware -- all but hardware has been tested (which *ought* to be entirely correct). Of course, all aspects of system-level correctness of the project can only be properly guessed at severely into post-implementation, which the time of the report does not afford (let it be pretence that this short documented period of user-interaction accounts for a much longer period of the system being in inert use).

User feedback (primarily Matthew on being walked through the GUI, but including also a more wide audience) raised the following points, of which, many should be acknowledged in subsequent maintenance. (It is important to recognise that the logbook prepended to this report describes -- although not very well -- ongoing consultation of end-users appropriate to the development of the GUI, and the client becomes explicitly involved again in the evaluation):

- Internal help screens would be appreciated (although the creation of a well-formatted external manual diminishes this necessity).

- Ability to re-order relations and symbols, or perhaps have symbols arranged numerically.

- Alternatives to the colour-coding system.

- GUI for saving functionality, rather than shortcut keys only.

- The removal of alert boxes as ways of conveying information, in favour of less intrusive ways.

4.6. CLI RUN-THROUGH

The GUI's functionality is built upon that of the CLI, so only features that work differently need to be tested. A sufficient manual run-through follows, where speculation that the CLI is correct (the author's scepticism is hushed by their undocumented prodding of the CLI).

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

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

1. Add a fact
```

```
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 2
Enter the fact number to delete:
>>> -1
Failed to delete fact: fact is out of range

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 2
Enter the fact number to delete:
>>> invalid
Could not delete fact.

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 2
Enter the fact number to delete:
>>> 0
Failed to delete fact: fact is out of range

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

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 3
There are no facts.

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 2
Enter the fact number to delete:
>>> 1
Failed to delete fact: fact is out of range

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][2]

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

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=0

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

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

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

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

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

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 1
Enter the fact to add:
>>> [2] = [3][4](1 - [5])

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 4
Enter the request:
>>> invalid
The request is malformed.

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

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 5
Did not recognise command.

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>> 9
Did not recognise command.
```

```
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 - [3])(1 - [4])(1 - [5])
0/0(1 - [0])(1 - [2])(1 - [3])(1 - [4])[5]
0/0(1 - [0])(1 - [2])(1 - [3])[4](1 - [5])
0/0(1 - [0])(1 - [2])(1 - [3])[4][5]
0/0(1 - [0])(1 - [2])[3](1 - [4])(1 - [5])
0/0(1 - [0])(1 - [2])[3](1 - [4])[5]
1/0(1 - [0])(1 - [2])[3][4](1 - [5])
0/0(1 - [0])(1 - [2])[3][4][5]
1/0(1 - [0])[2](1 - [3])(1 - [4])(1 - [5])
1/0(1 - [0])[2](1 - [3])(1 - [4])[5]
1/0(1 - [0])[2](1 - [3])[4](1 - [5])
1/0(1 - [0])[2](1 - [3])[4][5]
1/0(1 - [0])[2][3](1 - [4])(1 - [5])
1/0(1 - [0])[2][3](1 - [4])[5]
1/0(1 - [0])[2][3][4][5]
1/0[0](1 - [2])(1 - [3])(1 - [4])(1 - [5])
1/0[0](1 - [2])(1 - [3])(1 - [4])[5]
1/0[0](1 - [2])(1 - [3])[4](1 - [5])
1/0[0](1 - [2])(1 - [3])[4][5]
1/0[0](1 - [2])[3](1 - [4])(1 - [5])
1/0[0](1 - [2])[3](1 - [4])[5]
1/0[0](1 - [2])[3][4](1 - [5])
1/0[0](1 - [2])[3][4][5]
1/0[0][2](1 - [3])(1 - [4])(1 - [5])
1/0[0][2](1 - [3])(1 - [4])[5]
1/0[0][2](1 - [3])[4](1 - [5])
1/0[0][2](1 - [3])[4][5]
1/0[0][2][3](1 - [4])(1 - [5])
1/0[0][2][3](1 - [4])[5]
1[0][2][3][4](1 - [5])
1/0[0][2][3][4][5]

1. Add a fact
2. Delete a fact
3. Show a list of facts
4. Request a solution.
>>>
Press any key to continue . . .
```

5. OPERATIONAL ANALYSIS

5.1. ON PRECISION

The analysis of this program is an academic exercise above anything else, and as such, doesn't warrant much attention in any state of urgency. To develop mathematical models of the system would be, it is my regret to say, a task as enjoyable as its length. It is thus that the content under the next heading draws non-specific conclusions, and much evidence comes from the unshared experience of the writer and the reader.

5.2. ANALYSIS

It was initially hypothesised that parallelisation will have a significant impact on the performance of the program, and it was set as a success point in `III. 2.2. (iv) The main process of the P.L.E must be manually parallelisable. In the preliminary analysis (`III. 5.2.), the idea of parallelisation was further explored, and let it be recalled the assertion "… Whence, it can be deduced that the only scenario where optimisation is worth anything is when both sets of 2^n [both the act of calculating and printing constituents] can be reduced.". In the programs current domain of operation, the results are all printed to a single screen. How then, could the process of printing be parallelised? Any such act would lead to obvious bottlenecks, and the complexity of the problem retains its place as being exponential in time complexity. It is for this reason that the program is not parallelised.

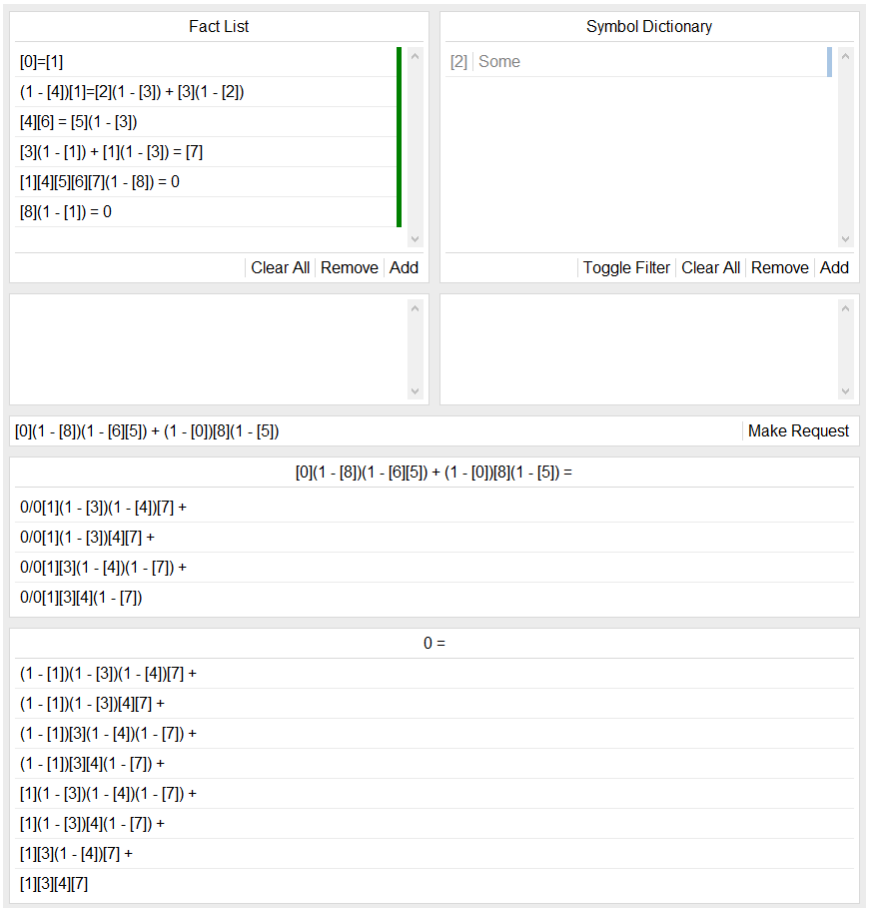
Let the internals of the system be now examined in a series of ad hoc white-box tests to get a feel for the inherent inefficiencies of the system. Intercepting a small random Relation in the mainline process (before a call to the insertArgument function) reveals the value:

```
(1)((0+(((0))(1-([1]))+([1])(1-((0))))+(((1-[4])[1])(1-((0)(1-[3])+[3](1-(0
))))+((0)(1-[3])+[3](1-(0)))(1-((1-[4])[1])))((1-(((0))(1-([1]))+([1])(1-((
0))))))+(([4][6])(1-([5](1-[3])))+([5](1-[3]))(1-([4][6])))(((1-(((0))(1-([
1]))+([1])(1-((0))))))(1-(((1-[4])[1])(1-((0)(1-[3])+[3](1-(0))))+((0)(1-[3
])+[3](1-(0)))(1-((1-[4])[1])))))+(([3](1-[1])+[1](1-[3]))(1-([7]))+([7])(1
-([3](1-[1])+[1](1-[3]))))((((1-(((0))(1-([1]))+([1])(1-((0))))))(1-(((1-[4
])[1])(1-((0)(1-[3])+[3](1-(0))))+((0)(1-[3])+[3](1-(0)))(1-((1-[4])[1])))))
)(1-(([4][6])(1-([5](1-[3])))+([5](1-[3]))(1-([4][6])))))))(0+(((0))(1-([1])
)+([1])(1-((0))))+(((1-[4])[1])(1-((1)(1-[3])+[3](1-(1))))+((1)(1-[3])+[3](
1-(1)))(1-((1-[4])[1])))((1-(((0))(1-([1]))+([1])(1-((0))))))+(([4][6])(1-(
[5](1-[3])))+([5](1-[3]))(1-([4][6])))(((1-(((0))(1-([1]))+([1])(1-((0)))))
)(1-(((1-[4])[1])(1-((1)(1-[3])+[3](1-(1))))+((1)(1-[3])+[3](1-(1)))(1-((1-
[4])[1])))))+(([3](1-[1])+[1](1-[3]))(1-([7]))+([7])(1-([3](1-[1])+[1](1-[3
])))))((((1-(((0))(1-([1]))+([1])(1-((0))))))(1-(((1-[4])[1])(1-((1)(1-[3])+
[3](1-(1))))+((1)(1-[3])+[3](1-(1)))(1-((1-[4])[1])))))(1-(([4][6])(1-([5](
1-[3])))+([5](1-[3]))(1-([4][6])))))))-((1)((0+(((1))(1-([1]))+([1])(1-((1)
)))+(((1-[4])[1])(1-((0)(1-[3])+[3](1-(0))))+((0)(1-[3])+[3](1-(0)))(1-((1-
[4])[1])))((1-(((1))(1-([1]))+([1])(1-((1))))))+(([4][6])(1-([5](1-[3])))+(
[5](1-[3]))(1-([4][6])))(((1-(((1))(1-([1]))+([1])(1-((1))))))(1-(((1-[4])[
1])(1-((0)(1-[3])+[3](1-(0))))+((0)(1-[3])+[3](1-(0)))(1-((1-[4])[1])))))+(
([3](1-[1])+[1](1-[3]))(1-([7]))+([7])(1-([3](1-[1])+[1](1-[3]))))((((1-(((
```

```
1))(1-([1]))+([1])(1-((1))))))(1-(((1-[4])[1])(1-((0)(1-[3])+[3](1-(0))))+(
(0)(1-[3])+[3](1-(0)))(1-((1-[4])[1])))))(1-(([4][6])(1-([5](1-[3])))+([5](
1-[3]))(1-([4][6])))))))(0+(((1))(1-([1]))+([1])(1-((1))))+(((1-[4])[1])(1-(
(1)(1-[3])+[3](1-(1))))+((1)(1-[3])+[3](1-(1)))(1-((1-[4])[1])))((1-(((1))(
1-([1]))+([1])(1-((1)))))+(([4][6])(1-([5](1-[3])))+([5](1-[3]))(1-([4][6]
)))(((1-(((1))(1-([1]))+([1])(1-((1))))))(1-(((1-[4])[1])(1-((1)(1-[3])+[3]
(1-(1))))+((1)(1-[3])+[3](1-(1)))(1-((1-[4])[1])))))+(([3](1-[1])+[1](1-[3]
))(1-([7]))+([7])(1-([3](1-[1])+[1](1-[3])))))(((1-(((1))(1-([1])))+([1])(1-
((1))))))(1-(((1-[4])[1])(1-((1)(1-[3])+[3](1-(1))))+((1)(1-[3])+[3](1-(1))
)(1-((1-[4])[1])))))(1-(([4][6])(1-([5](1-[3])))+([5](1-[3]))(1-([4][6]))))
))))
```

Looking at this, it is immediately apparent that there is a somewhat large amount of wasted space -- but perhaps less than initially expected. There is no meaningless whitespace, and the distribution of instances where easy simplification (such as changing "((0))" to "(0)" or "1-(0)" to "(1)") is sparse. The only real optimisation to Relation storage (which is the primary factor in determining the speed and memory usage of the program) would be to recognise more fundamental simplifications involving symbols -- rules must be recognised such as "δ(1 - [0]) + δ[0] = δ". These rules are greatly difficult to simplify according to, and verily, the performance at this degree of optimisation is surprisingly efficient.

A simple simulation now serves to test the real-time operability of the solution. It takes merely one second for the machine to process this input, giving the accompanying output:



And with such a benchmark in mind, it becomes plain to normal sight that the PLE is sufficiently fast for all practical purposes -- although, testing it with the kangaroo scenario (`V. 4.4.), which is of quite a high complexity, taking almost two minutes, reveals that the program becomes non-operational for sufficiently large amounts and complexities of inputs -- but there isn't much easy optimisation that can help performance in any way.

6. QUALITY (AN ASSESSMENT OF)

6.1. PROBLEM SPECIFICATION

Time affords no more extensions to the project, and thus, the quality of the solution can be finally assessed, with respect to the requirements stipulated throughout `II. (particularly, `II. 2.). Let it firstly be restated the boundaries:

(i) The Propositional Logic Engine (P.L.E) will not perform lexical analysis; all information must be inputted in mathematical form, and will be outputted also in mathematical form.

(ii) The P.L.E will not process any problems of quantity or probabilities: the only quantifiers the P.L.E will interpret are some, none, and all.

(iii) The P.L.E allows for the aided interpretation of a solution, it does not, itself, wholly provide the solution.

(iv) The P.L.E operates in the universe of propositional logic, that is, all results inferred from outputs or intermediary equations of the P.L.E will be equally as fallible as the premises combined, whence:

(v) The P.L.E will not yield objectively true solutions.

(vi) The P.L.E will not be able to derive implied relations between symbols. The system will only act on explicitly provided information, or information stored in a database.

These boundaries must be kept in mind through the rest of the evaluation. It must now be considered what was promised:

(i) The functionality of the P.L.E must be: given a relationship between symbols, accept and parse a request for knowledge on a subject, and return the knowledge the system has on the requested subject. The P.L.E must interface in this format of exchange:

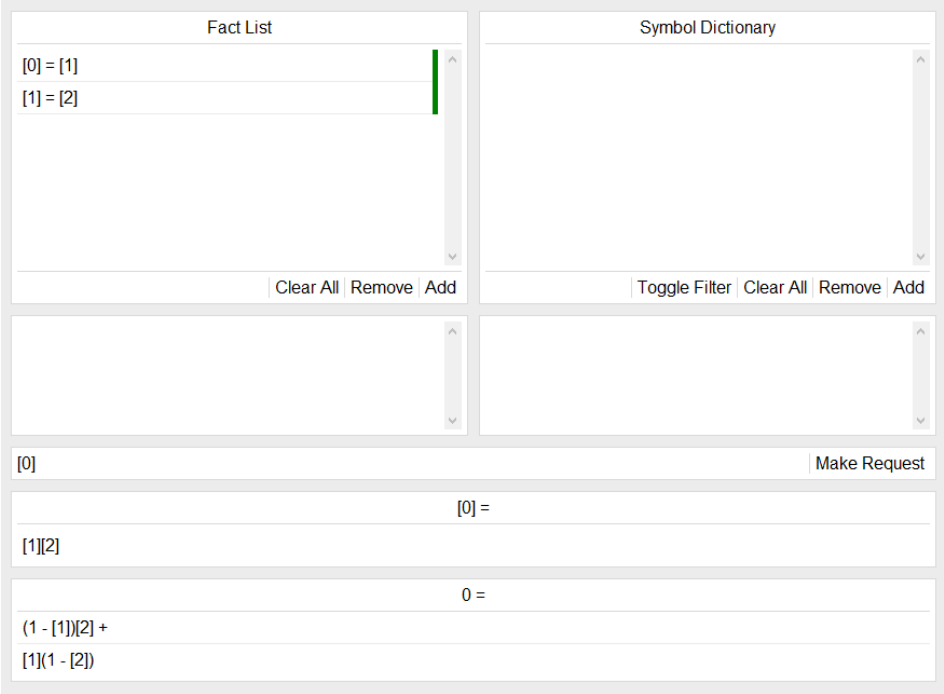
1. User says "*These* facts are true".

2. User asks "What is true about *this* subject?"

3. The P.L.E fulfils the request, saying "*These* facts are true", --

Input	Process	Output
Rules (relationships between symbols), Request for information on a subject	Conglomerate all given rules, then search and compile information relevant to the requested subject.	Information about the requested subject.

Let the truth of this success criterion be demonstrated by annotating the GUI:



This image represents the exchange format:

1. User says "[0] = [1] and [1] = [2] are true."

2. User asks "What is true about [0]?"

3. The P.L.E fulfils the request, saying "[0] = [1][2], and 0 = (1 - [1])[2] + [1](1 - [2])".

-- Which is in parity with what was required. The second criteria can now be considered.

(ii) The P.L.E must accept and parse multiple relations.

This feature is manifestly present, as demonstrated in the image above.

(iii) The P.L.E must have saving functionality for symbol's names and the relationships between symbols.

This feature, also, is present. Ctrl-s / ctrl-o allow the saving and loading of facts, descriptions of facts, symbol numbers, symbol names, and whether or not symbols are in superposition, which fits this requirement.

(iv) The main process of the P.L.E must be manually parallelisable.	REVOKED

This descriptor is not met, and such an occurrence was foreseen, and the criterion was modified with acceptance of the client (I. 1.34.).

(v) The P.L.E must conform reasonably to the following quality criteria by the perspective of all parties involved:

- correct and accurate
- reliable
- usable
- efficient
- secure

The developer holds that the correctness and accuracy has been verified by several system-level test cases, as well as their experience in testing the incomplete system. Reliability is verified likewise. To meet the usability requirement, it was the developer who designed (in the compilation of feedback) and implemented the interface, so by their knowledge of its intended workings, the system is perfectly usable provided the user has an irremovably necessary understanding of propositional logic. Usability can be further justified in the observations that the program is forgiving of mistakes (asking for confirmation before you clear symbols / facts), has its screen elements separated meaningfully, comes with an external manual, uses labels appropriately, and is appropriately simple.

Confidence is also placed by the developer in the efficiency of the system, although this has not been founded on any mathematical proof. In consideration of security, the developer understands the worst possible exploit of the system to be to a no more of a risk than visiting an untrusted website that has the ability to override or write .ple files. This risk is majorly thwarted by security measures, but is still irremovably present.

(vi) The P.L.E must conform to the following criteria by the developer's perspective:
- interoperable, flexible, and reusable
- testable
- maintainable
- possess high quality wholly

The P.L.E's core and interface are fundamentally separated, making the system easily modifiable to suit many different scenarios with minimal work. The core was designed to be testable, being built around unit tests (V. 3.), but the interfaces were not designed as such, and no debugging tools are provided for the GUI. It is best to consider the core and the interfaces separately, and acknowledge that the core functionality is easily testable, but the GUI requires human effort in testing in its present state. Likewise, it is felt that the core is of a high degree of code quality, and is easily maintainable thus, but the GUI doesn't achieve design perfection internally (see next section, V. 6.2.). Externally, the developer finds few faults in the GUI that they do not have full awareness of, and have neglected to repair in correlation with that described in the next section.

(vii) The adding of additional features, if it is to occur at all, will be done so in consultation with the client, and will have their own specifications developed externally to this section (perhaps tacitly).

Throughout the development process, constant contact has been kept with the client, and the addition and removal of several features has been discussed -- and it is thus that all of the criteria of the specification have been met, and the project is classifiable as a success within this domain.

6.2. PROJECT MANAGEMENT

The project, wholly, was successful, yet, many desirable features were not added. There follows a list of things I would like to do / would have liked to do, but didn't have time to do / won't have time to do:

- Implement ETA / loading screen. This was initially planned, and I tried to implement it -- but the PySide DOM and the Python engine live in entirely separate universes (asynchronously), so I cannot print to the DOM during mass processing. This similarly made it impossible to implement pause / cancel operability without a major reworking of the system, and it was decided that this be left out for simplicity (with the permission of the client), knowing well that such a feature was a major part of the planned solution.

- Rewrite GUI.html, runGUI.py, and runCLI.py. These are the files I feel are of markedly lower quality than the rest, and I lay my blame in my web-development habits. Particularly, I failed to associate JavaScript functions with objects, meaning that I was hesitant to create more functions, leading to longer and less-self explanatory functions. I also have never quite got the hang of interacting with the DOM, and in the future, I have mind to create a strict DOM interface.

- Do a proper operational analysis. These are very enjoyable, and allow a lot of useful information to be extracted from the program, but time simply did not afford this, and it was deemed as unnecessary. In accordance with an analysis, I would find much joy in optimising.

- Parallelisation. This dream was dropped quite early, and it is important to note that had I not dropped it so early, the project would have almost certainly been late. It is important to consider the exclusion of features -- taking parallelisation as exemplification -- not merely as failures (in design and planning), but as being successes in project management.

- Implement a system to simplify output. As is, it is possible to conjure scenarios whereby the PLE responds with output of (or of the nature) "[1] = [2] + (1 - [2])", which the system ought to be able to obviously recognise as being equivalent to "[1] = 1". Within the source code is a comment expressing my unhappiness, "the constituentList array can be abbreviated at this point by the general rule [0][1] + [0](1 - [1]) = [0], but it is not a feature required for full functionality". It ought to be fairly easy to implement, and as such, I knew / know that if I really wanted it / want it, I can throw it in during maintenance.

- Option to turn on / off implied relations. As is, the P.L.E will spew out a long string under the heading "0 = " which is only useful in easily identifiable circumstances. A simple checkbox somewhere that expresses "Do you want me [the P.L.E] to give you a list of things relevant to your request that cannot possibly exist" would be very welcome.

- Stop preventing drag events. My sentiments are echoed by this blog post (as it exists on 31/07/2016): http://www.quirksmode.org/blog/archives/2009/09/the_html5_drag.html. The issue is agitated by the fact that I am not using any ordinary browser, but rather, a terribly undocumented portion of PySide (or perhaps the documentation is just very hard to find).

- CLI support for quantum superpositions. Currently, superpositions must be manually spoofed in the CLI, and are not supported.

- Rewrite have saving / loading works, and include it in the CLI. Quite riskily, saving / loading uses node.innerHTML as an essential component of operation, but in the real world, such horror would not be allowed.

- Mouse accessible save-load functionality. It is a very lazy decision to force the user to press ctrl-s and ctrl-o to save and load databases, but this has not been rectified due to GUI design difficulties (I don't know how to insert two buttons into the current layout without significant thought).

- Add internal help screens.

And many, many others.

It can now be asked why things went wrong, to which, I answer, they really didn't. Things aren't particularly bad, and I feel usual artisanship for most parts of the project. But indeed, they could have been done much better. I have learnt very much about the management of software projects (and things in general), but it feels to be in hard-learned abstract lessons that cannot easily be instantiated on paper in succinct rules. These new strategies, perhaps, are made obvious to the reader by virtue of the report form, and, peradventure, the reader was able to anticipate my mistakes and shortcomings, recognising by my reactions how I have adapted, and they too are aware of what I have learnt.

6.3. CLIENT SIGN-OFF

MAINTENANCE

PART VI.

1. MAINTENANCE LOG

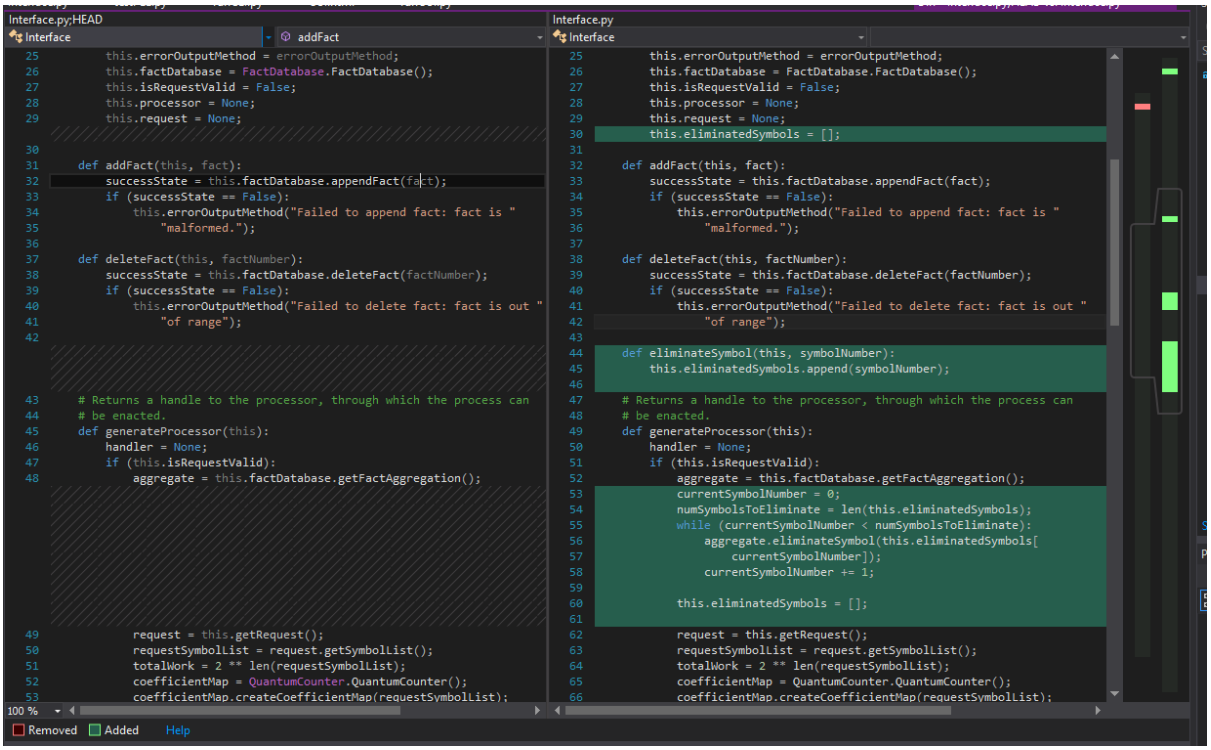
1.1. NECESSITY OF CHANGES AND LOGGING

With time, errors or unexpected behaviour that evades even the strictest test cases will emerge, or the needs of the end user will evolve or simply become more known. Perhaps, maintenance can be forced as a necessity by evolving hardware or newer software releases -- but the nature of the cause isn't of importance, rather, the fact that maintenance is invariably required in any software solution.

Required maintenance, if unlogged, becomes unwieldy to management. Version control quickly becomes an issue in unorganised software development (even pre-release). Perhaps developers who were not initially involved in the project must become familiar with the system, and without ample documentation of maintenance, their understanding of the system will certainly exist in discord with what actually happens to be. It is thus manifestly important that project maintenance is consistently logged within a workable system -- in this case, Visual Studio provides ample CASE tools.

1.2. SAMPLE LOG

Between the point of time in which some of the source code was copied into this document, and final release, a single modification to one of the already written-out files was enacted. Visual Studio beautifully summarises these changes:



It is thus that much of the process of log file creation is internalised developer-side. It serves best that still, in the event of actual modification, maintenance messages are appended to a list of version changes, and made available in ongoing documentation (generally online).

In accompaniment with full justification of these particular changes, let it be standardised that logs take the following form:

```
New Version: 1.0
Old Version: 0.0
Date: 27/07/2016
Author(s): Nicholas Killeen
Location(s): PLE.Interface.Interface.py
Purpose: Version 0.0 could not abstract away symbols -- namely, this
    version brings the system in line with the necessity of setting symbols
    into quantum states (whose mathematical values are 0/0), thus meaning
    that specific symbols can be filtered from the results of a request,
    and allowing for the otherwise disallowed nature of fact "all men are
    mortal".
Changes:
 > Interface.__init__(this, errorOutputMethod):
   - Now instantiates an empty member array this.eliminatedSymbols.
 > Interface.eliminateSymbol(this, symbolNumber):
   - This function has been added.
   - Appends symbolNumber to this.eliminatedSymbols.
 > Interface.generateProcessor(this):
   - After defining aggregate, eliminates each symbol from
     this.eliminatedSymbols, and then empties the array.
```

1.3. ACTUAL PSEUDO-MAINTENANCE

Testing (`V. 4.3.) revealed several errors of moderate priority. At this point, the source does has not been released (this can still be classified as iterative debugging), so the creation of actual maintenance logs is impossible, since there is no previous version to compare the modifications with. Still, it must be stipulated how these errors were rectified.

Problem	Solution
In symbol-number fields, pasting strings that contain new-lines cause empty text-nodes to be created within the contenteditable div, causing the symbol-number field to exceed its standard height of 1em, and overflow adjacent elements. Likewise when all characters of the field are dragged from the node.	All drag events are now being ignored "drop events must specifically be ignored due to their shoddy standards and implementation", and paste events specific to .symbolNumber nodes are ignored.
Certain relations that ought to be easily recognisable as INVALID (particularly, those with mismatched square brackets), are coloured as UNCHECKED by the input validator, and raise non-terminal errors in Python.	Completely changed the way that validity is checked: it is now checked in two stages, the first being _isRelationPseudoValid, which checkes for validity in all areas but that the relation exists in the Boolean domain {0, 1}. The second stage of testing is to determine the domain. This allows it to be calculated whether or not the symbol list can be correctly read from a Relation class. This implementation could conceivably change the way the PLE works, so checks 17, 18, and 19 (from `V. 4.3.) were repeated.

High symbol values, when inputted into relation fields, cause significant responsiveness issues when their validity is automatically checked.	A condition "max(symbolList) > LARGEST_SYMBOL" has been added to the Bridge.isValid function. Checks 17, 18, and 19 were repeated again.

There have also been additional changes to how new-lines work -- now, on the checking of fact validity, new-lines are iteratively replaced with empty space.

APPENDIX

PART VII.

1. SNIPPETS

1.1. EVAL.CPP

```
// eval.cpp
// Nicholas Killeen
// 25th January 2016
// Program to evaluate a string as an arithmetic equation,--
// without division functionality

#include "stdafx.h"
#include "stdlib.h"
#include "assert.h"
#include <iostream>

void runTests(void);

int parseDigit(char character);
int evaluateAdditionSubtraction(char data[], int length);
int getMaxDepth(char data[], int length);
char stringifyDigit(int value);
int power(int base, int exponent);
char* $stringify(int value, int length);
char* $evaluateInnerBracket(char data[], int length);
char* $evaluateInnerExpression(char data[], int length);
bool isStringDisparity(char string1[], char string2[], int length);
    // unused
bool areBrackets(char data[], int length);
bool hasEvaluableBrackets(char data[], int length);
bool hasEvaluableMultiplication(char data[], int length);
int eval(char data[], int length);

int main(int argc, char* argv[])
{
    runTests();
    return EXIT_SUCCESS;
}

void runTests(void)
{
    assert(parseDigit('0') == 0);
    assert(parseDigit('1') == 1);
    assert(parseDigit('2') == 2);
    assert(parseDigit('3') == 3);
    assert(parseDigit('4') == 4);
    assert(parseDigit('5') == 5);
    assert(parseDigit('6') == 6);
    assert(parseDigit('7') == 7);
    assert(parseDigit('8') == 8);
    assert(parseDigit('9') == 9);
    assert(parseDigit('c') == -1);
    assert(parseDigit('d') == -1);
```

```
    assert(parseDigit('[') == -1);
    assert(parseDigit(' ') == -1);

    assert(evaluateAdditionSubtraction("1", 1) == 1);
    assert(evaluateAdditionSubtraction("19", 2) == 19);
    assert(evaluateAdditionSubtraction("519586 ", 7) == 519586);
    assert(evaluateAdditionSubtraction("1+1", 3) == 2);
    assert(evaluateAdditionSubtraction("1+2+1+0", 7) == 4);
    assert(evaluateAdditionSubtraction("1-2+1-0", 7) == 0);
    assert(evaluateAdditionSubtraction("19+12-13", 8) == 18);
    assert(evaluateAdditionSubtraction(" 0 + 112", 8) == 112);

    assert(getMaxDepth("dogs", 4) == 0);
    assert(getMaxDepth("(cat)", 5) == 1);
    assert(getMaxDepth("((1))", 5) == 2);
    assert(getMaxDepth("((1)())", 7) == 2);
    assert(getMaxDepth("( 1(() )3(a))", 13) == 3);
    assert(getMaxDepth("(cats)   and ()((()()(((1 ) ()()(       dogs)()(
&()(((((1)()423()()6(3)()))))3)()()(
pigs)()()))(123())()()()))()(35)()()()(()1()()())",
        136) == 10);
    assert(getMaxDepth("((((((()))))))()", 15) == 6);
    assert(getMaxDepth("()((((((()))))))", 15) == 6);

    assert(stringifyDigit(0) == '0');
    assert(stringifyDigit(1) == '1');
    assert(stringifyDigit(2) == '2');
    assert(stringifyDigit(3) == '3');
    assert(stringifyDigit(4) == '4');
    assert(stringifyDigit(5) == '5');
    assert(stringifyDigit(6) == '6');
    assert(stringifyDigit(7) == '7');
    assert(stringifyDigit(8) == '8');
    assert(stringifyDigit(9) == '9');
    assert(stringifyDigit(10) == ' ');
    assert(stringifyDigit(-1) == ' ');

    assert(power(0, 1) == 0);
    assert(power(0, 5) == 0);
    assert(power(3, 0) == 1);
    assert(power(2, 0) == 1);
    assert(power(2, 1) == 2);
    assert(power(2, 2) == 4);
    assert(power(2, 10) == 1024);
    assert(power(5, 3) == 125);
    assert(power(10, 1) == 10);
    assert(power(10, 3) == 1000);
    assert(power(10, 7) == 10000000);

    char* $string1 = $stringify(195, 5);
    assert($string1[0] == '0');
    assert($string1[1] == '0');
    assert($string1[2] == '1');
```

```
    assert($string1[3] == '9');
    assert($string1[4] == '5');
    delete[] $string1;
    $string1 = NULL;

    char* $string2 = $stringify(0, 2);
    assert($string2[0] == '0');
    assert($string2[1] == '0');
    delete[] $string2;
    $string2 = NULL;

    char* $string3 = $stringify(-120, 5);
    assert($string3[0] == '-');
    assert($string3[1] == '0');
    assert($string3[2] == '1');
    assert($string3[3] == '2');
    assert($string3[4] == '0');
    delete[] $string3;
    $string3 = NULL;

    char* $string4 = $stringify(-120, 6);
    assert(evaluateAdditionSubtraction($string4, 6) == -120);
    delete[] $string4;
    $string4 = NULL;

    char* $string5 = $stringify(0, 1);
    assert(evaluateAdditionSubtraction($string5, 1) == 0);
    delete[] $string5;
    $string5 = NULL;

    char* $string6 = $stringify(9134520, 8);
    assert(evaluateAdditionSubtraction($string6, 8) == 9134520);
    delete[] $string6;
    $string6 = NULL;

    char* $string7 = $stringify(-9134520, 8);
    assert(evaluateAdditionSubtraction($string7, 8) == -9134520);
    delete[] $string7;
    $string7 = NULL;

    char *$data1 = $evaluateInnerBracket("2*(2+1)", 7);
    assert($data1[0] == '2');
    assert($data1[1] == '*');
    assert($data1[2] == '0');
    assert($data1[3] == '0');
    assert($data1[4] == '0');
    assert($data1[5] == '0');
    assert($data1[6] == '3');
    delete[] $data1;
    $data1 = NULL;

    char *$data2 = $evaluateInnerBracket("31*(2-2)", 8);
    assert($data2[0] == '3');
```

```
    assert($data2[1] == '1');
    assert($data2[2] == '*');
    assert($data2[3] == '0');
    assert($data2[4] == '0');
    assert($data2[5] == '0');
    assert($data2[6] == '0');
    assert($data2[7] == '0');
    delete[] $data2;
    $data2 = NULL;

    char *$data3 = $evaluateInnerBracket("31*(2-5)", 8);
    assert($data3[0] == '3');
    assert($data3[1] == '1');
    assert($data3[2] == '*');
    assert($data3[3] == '-');
    assert($data3[4] == '0');
    assert($data3[5] == '0');
    assert($data3[6] == '0');
    assert($data3[7] == '3');
    delete[] $data3;
    $data3 = NULL;

    char *$exp1 = $evaluateInnerExpression("001*001", 7);
    assert($exp1[0] == '0');
    assert($exp1[1] == '0');
    assert($exp1[2] == '0');
    assert($exp1[3] == '0');
    assert($exp1[4] == '0');
    assert($exp1[5] == '0');
    assert($exp1[6] == '1');
    delete[] $exp1;
    $exp1 = NULL;

    char *$exp2 = $evaluateInnerExpression("002*010", 7);
    assert($exp2[0] == '0');
    assert($exp2[1] == '0');
    assert($exp2[2] == '0');
    assert($exp2[3] == '0');
    assert($exp2[4] == '0');
    assert($exp2[5] == '2');
    assert($exp2[6] == '0');
    delete[] $exp2;
    $exp2 = NULL;

    char *$exp3 = $evaluateInnerExpression("13*19", 5);
    assert(evaluateAdditionSubtraction($exp3, 5) == 247);
    delete[] $exp3;
    $exp3 = NULL;

    char *$exp4 = $evaluateInnerExpression("-99*99", 6);
    assert(evaluateAdditionSubtraction($exp4, 6) == -9801);
    delete[] $exp4;
    $exp4 = NULL;
```

```
    char *$exp5 = $evaluateInnerExpression("-99*-99", 7);
    assert(evaluateAdditionSubtraction($exp5, 7) == 9801);
    delete[] $exp5;
    $exp5 = NULL;

    char *$exp6 = $evaluateInnerExpression("0*0", 3);
    assert(evaluateAdditionSubtraction($exp6, 3) == 0);
    delete[] $exp6;
    $exp6 = NULL;

    char *$exp7 = $evaluateInnerExpression("0*-1", 4);
    assert(evaluateAdditionSubtraction($exp7, 4) == 0);
    delete[] $exp7;
    $exp7 = NULL;

    char *$exp8 = $evaluateInnerExpression("13+(990*156)", 12);
    assert($exp8[0] == '1');
    assert($exp8[1] == '3');
    assert($exp8[2] == '+');
    assert($exp8[3] == '(');
    assert($exp8[4] == '0');
    assert($exp8[5] == '1');
    assert($exp8[6] == '5');
    assert($exp8[7] == '4');
    assert($exp8[8] == '4');
    assert($exp8[9] == '4');
    assert($exp8[10] == '0');
    assert($exp8[11] == ')');
    delete[] $exp8;
    $exp8 = NULL;

    assert(isStringDisparity("ab", "ab", 2) == false);
    assert(isStringDisparity("ab", "ac", 2) == true);
    assert(isStringDisparity("A7", "a7", 2) == true);
    assert(isStringDisparity("a", "b", 1) == true);
    assert(isStringDisparity("5", "5", 1) == false);
    assert(isStringDisparity("otksa", "otksa", 5) == false);
    assert(isStringDisparity("otksa", "otksy", 5) == true);

    assert(areBrackets("()", 2) == true);
    assert(areBrackets("dog", 3) == false);
    assert(areBrackets("()()", 4) == true);
    assert(areBrackets("ab()", 4) == true);

    assert(hasEvaluableMultiplication("12", 2) == false);
    assert(hasEvaluableMultiplication("1)*(2", 4) == false);
    assert(hasEvaluableMultiplication("1a*(2", 4) == false);
    assert(hasEvaluableMultiplication("1)*b2", 4) == false);
    assert(hasEvaluableMultiplication("1)a(2", 4) == false);
    assert(hasEvaluableMultiplication("1*2", 4) == true);

    assert(hasEvaluableBrackets("(1+1)", 5) == true);
```

```
    assert(hasEvaluableBrackets("12(1-1)", 7) == true);
    assert(hasEvaluableBrackets("12(151)", 7) == true);
    assert(hasEvaluableBrackets("(12-1)+(51)", 11) == true);
    assert(hasEvaluableBrackets("412*1+517", 9) == false);
    assert(hasEvaluableBrackets("(12+2*2)", 8) == false);

    assert(eval("1 + 12*(13 + 1)", 15) == 169);
    assert(eval("1 - 12*(13 - 1)*2", 17) == -287);
    assert(eval("(1) + (-1)", 10) == 0);
    assert(eval("1", 1) == 1);
    assert(eval("(12)", 4) == 12);
    assert(eval("1 + 2 + 3 - 4", 13) == 2);
    assert(eval("11*(0)", 6) == 0);
    assert(eval("22*(20)", 7) == 440);
    assert(eval("02*(02)", 7) == 4);
    assert(eval("(12)*02", 7) == 24);
    assert(eval("(1)*(1)*(1)+(1-1)*(1)+1", 23) == 2);
    assert(eval("((1))", 6) == 1);
    assert(eval("((1 + 1*(12 - 4))*2)", 20) == 18);

    std::cout << "All tests passed!!!" << std::endl;
    std::cout << "You are awesome!" << std::endl;
}

int parseDigit(char character)
{
    // this can be optimised to perform in constant time by using ASCII
    int value;
    if (character == '0')
    {
        value = 0;
    }
    else if (character == '1')
    {
        value = 1;
    }
    else if (character == '2')
    {
        value = 2;
    }
    else if (character == '3')
    {
        value = 3;
    }
    else if (character == '4')
    {
        value = 4;
    }
    else if (character == '5')
    {
        value = 5;
    }
    else if (character == '6')
```

```
    {
        value = 6;
    }
    else if (character == '7')
    {
        value = 7;
    }
    else if (character == '8')
    {
        value = 8;
    }
    else if (character == '9')
    {
        value = 9;
    }
    else
    {
        value = -1;
    }
    return value;
}

int evaluateAdditionSubtraction(char data[], int length)
{
    int sum = 0;
    char currentOperator = '+';
    for (int index = 0; index < length; ++index)
    {
        if (data[index] == ' ')
        {
            // do nothing ...
        }
        else if (parseDigit(data[index]) != -1)
        {
            int value = 0;
            while ((parseDigit(data[index]) != -1) && index < length)
            {
                value *= 10;
                value += parseDigit(data[index]);
                index += 1;
            }
            --index;
            // we overstepped to check that the character ahead isn't a
            // digit

            if (currentOperator == '+')
            {
                sum += value;
            }
            else if (currentOperator == '-')
            {
                sum -= value;
            }
```

```
            else
            {
                throw;
            }
        }
        else
        {
            currentOperator = data[index];
        }
    }
    return sum;
}

int getMaxDepth(char data[], int length)
{
    int maxDepth = 0;
    int depth = 0;
    for (int index = 0; index < length; ++index)
    {
        if (data[index] == '(')
        {
            ++depth;
        }
        else if (data[index] == ')')
        {
            --depth;
        }

        if (depth > maxDepth)
        {
            ++maxDepth;
        }
    }
    return maxDepth;
}

char stringifyDigit(int value)
{
    char character = ' ';
    if (value == 0)
    {
        character = '0';
    }
    else if(value == 1)
    {
        character = '1';
    }
    else if (value == 2)
    {
        character = '2';
    }
    else if (value == 3)
    {
```

```
        character = '3';
    }
    else if (value == 4)
    {
        character = '4';
    }
    else if (value == 5)
    {
        character = '5';
    }
    else if (value == 6)
    {
        character = '6';
    }
    else if (value == 7)
    {
        character = '7';
    }
    else if (value == 8)
    {
        character = '8';
    }
    else if (value == 9)
    {
        character = '9';
    }
    return character;
}

int power(int base, int exponent)
{
    int compound = 1;
    for (int iii = 0; iii < exponent; ++iii)
    {
        compound *= base;
    }
    return compound;
}

char* $stringify(int value, int length)
{
    char *$expression = new char[length];
    if (value < 0)
    {
        $expression[0] = '-';
    }
    else
    {
        $expression[0] = '0';
    }

    int unexpressedValue = value;
    if (unexpressedValue < 0)
```

```
    {
        unexpressedValue *= -1;
    }
    for (int index = 1; index < length; ++index)
    {
        int digit = unexpressedValue;
        unexpressedValue -= unexpressedValue % (power(10, index));
        digit -= unexpressedValue;
        digit /= power(10, index - 1);
        char character = stringifyDigit(digit);

        $expression[length - index] = character;
    }

    return $expression;
}

char* $evaluateInnerBracket(char data[], int length)
{
    // 1. find the start and end indexes of the inner most bracket set.
    int maxDepth = getMaxDepth(data, length);
    int depth = 0;
    int index = 0;
    while (depth < maxDepth)
    {
        assert(index < length);
        if (data[index] == '(')
        {
            ++depth;
        }
        else if (data[index] == ')')
        {
            --depth;
        }
        ++index;
    }
    int startIndex = index;
    int endIndex = startIndex;
    char lastCharacter = ' ';
    while (lastCharacter != ')')
    {
        assert(endIndex < length);
        lastCharacter = data[endIndex];
        ++endIndex;
    }

    // 2. evaluate the expression within the brackets
    int expressionLength = endIndex - startIndex - 1;
    // alloc
    char *expression = new char[expressionLength];
    for (index = startIndex; index < endIndex - 1; ++index)
    {
        expression[index - startIndex] = data[index];
```

```
    }
    int value = evaluateAdditionSubtraction(expression, expressionLength);
    // dealloc
    delete[] expression;
    expression = NULL;

    // 3. rewrite the expression into newData without the bracket.
    // alloc -- DEALLOC IS DELEGATED TO CALLER
    char *newData = new char[length];
    for (index = 0; index < length; ++index)
    {
        if (index > startIndex - 2 && index < endIndex)
        {
            newData[index] = ' ';
        }
        else
        {
            newData[index] = data[index];
        }
    }

    // 4. write the value into newData in the white space where the bracket
    // was.
    // alloc
    char* $evaluatedExpression = $stringify(value, expressionLength + 2);

    for (index = startIndex - 1; index < endIndex; ++index)
    {
        newData[index] = $evaluatedExpression[index - startIndex + 1];
    }

    // dealloc
    delete[] $evaluatedExpression;

    return newData;
}

char* $evaluateInnerExpression(char data[], int length)
{
    int expressionAddress;
    int index = 0;
    while (index < length)
    {
        assert(index < length);
        if (data[index] != ')' && data[index + 1] == '*' && data[index + 2]
            != '(')
        {
            expressionAddress = index + 1;
        }
        ++index;
    }

    int LHSIndex = expressionAddress;
```

```
    do
    {
        --LHSIndex;
    } while (LHSIndex > 0 && (data[LHSIndex - 1] == '-' ||
        parseDigit(data[LHSIndex - 1]) != -1));

    int RHSIndex = expressionAddress;
    do
    {
        ++RHSIndex;
    } while (RHSIndex < length - 1 && parseDigit(data[RHSIndex + 1]) !=
        -1);

    int LHSStart = LHSIndex;
    int LHSLength = expressionAddress - LHSIndex;
    int RHSStart = expressionAddress + 1;
    int RHSLength = RHSIndex + 1 - RHSStart;

    // alloc
    char *LHS = new char[LHSLength];

    for (index = 0; index < LHSLength; ++index)
    {
        LHS[index] = data[LHSStart + index];
    }

    int LHSVal = evaluateAdditionSubtraction(LHS, LHSLength);

    // dealloc
    delete[] LHS;
    LHS = NULL;

    // alloc
    char *RHS = new char[RHSLength];

    for (index = 0; index < RHSLength; ++index)
    {
        RHS[index] = data[RHSStart + index];
    }

    int RHSVal = evaluateAdditionSubtraction(RHS, RHSLength);
    // dealloc
    delete[] RHS;
    RHS = NULL;

    int result = LHSVal * RHSVal;
    // alloc
    char* $stringResult = $stringify(result, LHSLength + RHSLength + 1);

    // alloc
    char *$newData= new char[length];

    for (index = 0; index < length; ++index)
```

```
    {
        $newData[index] = data[index];
    }

    for (index = 0; index < LHSLength + RHSLength + 1; ++index)
    {
        $newData[LHSStart + index] = $stringResult[index];
    }

    // dealloc
    delete[] $stringResult;
    $stringResult = NULL;

    return $newData;
}

bool isStringDisparity(char string1[], char string2[], int length)
{
    bool hasEncounteredDisparity = false;
    for (int index = 0; index < length && !hasEncounteredDisparity;
        ++index)
    {
        hasEncounteredDisparity |= (string1[index] != string2[index]);
    }

    return hasEncounteredDisparity;
}

bool areBrackets(char data[], int length)
{
    bool hasFoundBracket = false;
    for (int index = 0; index < length; ++index)
    {
        hasFoundBracket |= (data[index] == '(');
    }
    return hasFoundBracket;
}

bool hasEvaluableMultiplication(char data[], int length)
{
    bool hasFoundEvaluableMultiplication = false;
    for (int index = 0; index < length - 2; ++index)
    {
        hasFoundEvaluableMultiplication |= (data[index] != ')' &&
            data[index + 1] == '*' && data[index + 2] != '(');
    }
    return hasFoundEvaluableMultiplication;
}

bool hasEvaluableBrackets(char data[], int length)
{
    bool hasEvaluableBrackets = areBrackets(data, length) &&
        !hasEvaluableMultiplication(data, length);
```

```
    return hasEvaluableBrackets;
}

int eval(char data[], int length)
{
    char *newData = new char[length];

    for (int index = 0; index < length; ++index)
    {
        newData[index] = data[index];
    }
    while (areBrackets(newData, length))
    {
        while (hasEvaluableBrackets(newData, length))
        {
            char *$intermediaryData = $evaluateInnerBracket(newData,
                length);
            delete[] newData;
            newData = $intermediaryData;
            $intermediaryData = NULL;
        }

        while (hasEvaluableMultiplication(newData, length))
        {
            char *$intermediaryData = $evaluateInnerExpression(newData,
                length);
            delete[] newData;
            newData = $intermediaryData;
            $intermediaryData = NULL;
        }
    }

    int evalResult = evaluateAdditionSubtraction(newData, length);
    delete[] newData;
    newData = NULL;

    return evalResult;
}
```