



Refining the Logic Analyser

Pravar Seth

MInf Project (Part 2) Report

Master of Informatics
School of Informatics
University of Edinburgh

2021

Abstract

This a report for the second part of a two year project. This report describes the scope of using a teaching to help the students of Informatics 1A. It firstly gives a background on the content, the existing tools and research. It then explains the work involved in part 2 of the project. The report then evaluates the tool with unit testing and also user testing.

Acknowledgements

I would like to thank my supervisor Michael Fourman

Contents

1	Introduction	1
1.1	The Problem	1
1.2	Preliminaries	1
1.3	Overview of the current tool	4
1.4	The Environment	5
1.5	Aim of the tool	7
2	Background	8
2.1	Overview of Part 1	8
2.1.1	Summary of work done in Part 1	8
2.1.2	Limitations of Part 1	8
2.1.3	Further Work discussed in Part 1	9
2.2	Existing Tools	10
2.2.1	Truth Table Generator - Stanford University	10
2.2.2	Truth Table Creator	11
2.2.3	Logic calculator: Server-side Processing	12
2.3	Existing Research	15
3	Refining the Logic Analyser	16
3.1	Summary of work done in Part 2	16
3.2	Project Restructure	17
3.3	Redesigning the Random expression generator	17
3.4	Additional syntax and operators	19
3.5	Redesigning the User Interface	20
3.6	Redesign of user and question data storage	21
3.7	Redesign of CNF Minimisation Function	21
3.8	Improving User Interactivity	22
3.9	Addition of Sub-Questions	23
3.10	Truth-Table Questions	24
3.11	The Addition of Karnaugh Maps	25
3.12	Addition of DNF conversion	26
4	Evaluation	28
4.1	Unit Testing	28
4.1.1	Unit Testing Framework	28
4.1.2	Random Testing	28

4.1.3	White-box Testing	29
4.2	User Testing	30
4.2.1	Course Testing	30
4.2.2	Think Aloud Study	32
4.3	Project Completion	34
	Bibliography	35
	Appendices	36
	A Think Aloud study Questions/Tasks	37

Chapter 1

Introduction

1.1 The Problem

Informatics 1A, an introductory course in Informatics, is studied by Informatics students in their first semester of their undergraduate degree. A major part of the course is Computation and Logic.

The higher prevalence of online teaching poses many challenges for students. Challenges include: many students are not available for live teaching depending on their time zone, resources of tutors can be very limited, and many students may prefer independent learning in their own convenient time.

Computation and Logic is an important subject area that can take advantage of online interactivity. Due to the nature of the subject, students can often learn more efficiently from examples rather than reading or watching lectures. This gives motivation for an online teaching tool which as well as providing many examples can also interact with the students and walk through the examples to ensure the student can fully understand the course content.

A major part of Computation and Logic is Boolean logic. Students may struggle to fully understand Boolean logic without actively engaging with examples. Therefore, there is large scope for an online tool to assist students in this area.

1.2 Preliminaries

This project assumes basic knowledge of the Computation and Logic section of the Informatics 1A course [1]

The key parts of understanding are:

1. Boolean Algebra: Atoms and Boolean Expressions
2. Truth Tables

3. Boolean operators: 'NOT', 'AND', 'OR', 'XOR', 'Single Implications', 'Double Implications' and 'Ternary operators'
4. CNF (Conjunctive Normal Form) / DNF (Disjunctive Normal Form)
5. Karnaugh Maps

An atom is a Boolean object which has two values. For simplicity, this project will use the values 0 and 1 to represent the two values. In this project, we can use any non-white space characters to represent atoms so long as the characters used are not reserved for other operators.

A literal can be either an atom or a negated atom. A negated atom has an opposing value of the atom. For example: NOT A has the value of 1 if A is 0 and 0 if A is 1.

A Boolean expression is either a literal or a structure formed by combining sub-expressions or literals together using Boolean operators. Such an expression has an overall value of 0 or 1 depending on the values of all the atoms it contains. For example: 'A AND B' is a Boolean expression that is made up of two literals: 'A' and 'B' and a Boolean operator 'AND'. In this example, the expression has the overall value of 1 only when 'A' and 'B' are both 1 and it will have the value of 0 otherwise.

A Boolean expression has a corresponding truth table. A truth table tabulates the value of the expression for every possible combination of atom values. For each combination, a corresponding overall value for the expression is shown. Examples can be seen in Figure 1.1 with the different Boolean operators that are recognised in this project.

A	$\neg A$	A	B	$A \wedge B$	A	B	$A \vee B$	A	B	$A \oplus B$	A	B	$A \rightarrow B$
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	1	0	0	1	1	0	1	1	0	1	1
1	0	1	0	0	1	0	1	1	0	1	1	0	0
		1	1	1	1	1	1	1	1	0	1	1	1

A	B	C	$A ? B : C$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

A	B	$A \leftrightarrow B$
0	0	1
0	1	0
1	0	0
1	1	1

Figure 1.1: Truth Tables for each Boolean operator

A variety of different syntaxes are used for the Boolean operations — Figure 1.2 shows the examples considered in this project. Note that the default syntax is what is formally used in Informatics 1A course.

Operator	Default Syntax	Other Possible Syntaxes
NOT	\neg	NOT not - \sim !
AND	\wedge	AND and . & && /\ *
XOR	\oplus	XOR xor
OR	\vee	OR or + \/
Single Implication	\rightarrow	-> =>
Double Implication	\leftrightarrow	<-> <=>
Ternary Operator	? :	

Figure 1.2: Different syntaxes for each Boolean operator

Many different expressions can produce the same truth table. These expressions can be considered equivalent since they evaluate to the same value under all conditions. Therefore, there are standardised forms of how an equivalent expression can be formed.

A conjunction of sub-expressions is the combination of sub-expressions using the 'AND' operator, and similarly a disjunction of sub-expressions is the combination of sub-expressions using the 'OR' operator.

Expressions are in CNF(Conjunction Normal Form) when the expression is either: a conjunction of a (disjunction of literals) or a conjunction of literals, and similarly expressions are in DNF(Disjunctive Normal Form) when the expression is either: a disjunction of a (conjunction of literals) or disjunction of literals.

For example: $(C \vee \neg D \vee \neg Y) \wedge (C \vee \neg D \vee G)$ is a CNF expression

whereas $\neg D \vee C \vee (G \wedge \neg Y)$ is a DNF expression

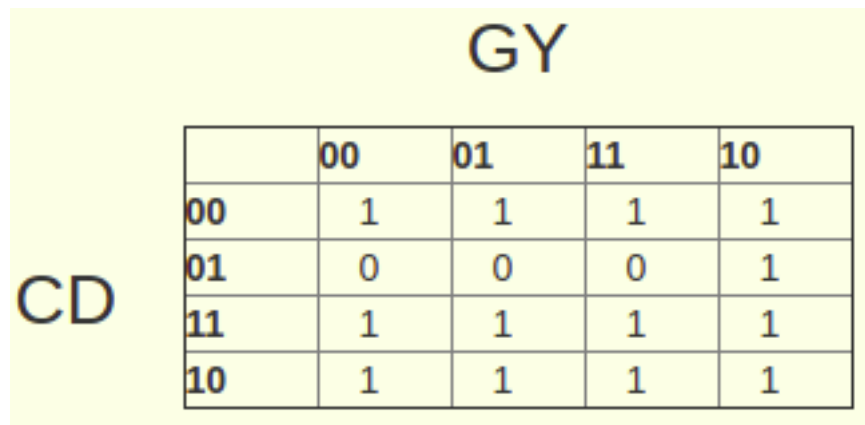
Note: Every Boolean expression has an equivalent form in CNF and DNF

Karnaugh Maps tabulates the value of the expression for every possible combination of atom values but in a different structure than compared to a truth table. An example can be seen in Figure 1.3.

The bold values indicate the possible combinations of the atoms. In this example, there are row keys: 00, 01, 11, 10 for CD and column keys: 00, 01, 11, 10 for GY.

The results are indicated by non-bold values of either 0 or 1. We can see the top left result is 1 indicating the expression evaluates to 1 when C and D are 0 and 0 respectively; and also G and Y are 0 and 0.

The key part of Karnaugh Maps, it allows students to visualise groups of 1s and 0s. A group of 1s can represent a conjunction of literals and multiple groups combined will therefore form a DNF expression.



		00	01	11	10
00		1	1	1	1
01		0	0	0	1
11		1	1	1	1
10		1	1	1	1

Figure 1.3: Karnaugh Map of $\neg D \vee C \vee (G \wedge \neg Y)$

1.3 Overview of the current tool

This project focuses on the development of a tool for analysing Boolean expressions. The tool can be found on <http://pravar.pythonanywhere.com> or it can be run locally with the source code attached to this project.

This is an online interactive tool with two main modes:

1. Analyse Expressions: where a user can enter a Boolean expression or generate one randomly. The user can then retrieve a corresponding Truth Table, Karnaugh Map or the steps to convert the expression to CNF(Conjunctive Normal Form) or DNF(Disjunctive Normal Form).
2. Answer Questions: where a user can answer questions with five different difficulties. There are unlimited questions which are randomly generated. Possible questions include: perform a manipulation to an expression, convert an expression to CNF/DNF, construct a Boolean expression that satisfies a given Truth Table or a given Karnaugh Map. The tool can then generate feedback on the student's answer or generate a sample solution.

Figures 1.4 and 1.5 shows the two main modes.

Logic Analyser Home **Analyse Expressions** Answer Questions Inf1a Course Page

Enter your Expression

<u>Operator</u>	<u>Precedence</u>	<u>Accepted Syntax</u>
NOT	1	NOT not \neg - ~ !
AND	2	AND and \wedge . & & \wedge *
XOR	3	XOR xor \oplus
OR	4	OR or \vee + \vee
Implication	5	\rightarrow -> =>
Double Implication	6	\leftrightarrow <-> <=>
Ternary Operator	7	? :

For example: (a or b) and (c -> d)

Display Truth Table
Display Karnaugh Map
Generate Random Expression
Convert to CNF
Convert to DNF
Return to Homepage

Don't forget to submit feedback by clicking here

Figure 1.4: Analyse Expressions Page

Logic Analyser Home Analyse Expressions **Answer Questions** Inf1a Course Page

Q1.1 Form an expression that satisfies this Karnaugh Map:

CD

	00	01	11	10	
AB	00	1	1	0	0
01	1	1	0	1	
11	1	1	0	0	
10	1	1	0	0	

<u>Operator</u>	<u>Precedence</u>	<u>Accepted Syntax</u>
NOT	1	NOT not \neg - ~ !
AND	2	AND and \wedge . & & \wedge *
XOR	3	XOR xor \oplus
OR	4	OR or \vee + \vee
Implication	5	\rightarrow -> =>
Double Implication	6	\leftrightarrow <-> <=>
Ternary Operator	7	? :

For example: (a or b) and (c -> d)

Enter
See sample solution
Next Question
Change Difficulty
Return to Homepage

Don't forget to submit feedback by clicking here

Figure 1.5: Questions Page (currently on Medium difficulty)

1.4 The Environment

The tool is developed using Flask, Bootstrap, HTML and Python 3.8.

The tool can be run locally or through the host <http://pravar.pythonanywhere.com>

Figure 1.6 shows the current project directory, the instructions to run the tool locally are included in the 'README.txt' file.

```
Logic Analyser
├── templates
│   ├── expressionAnalyser.html
│   ├── home.html
│   ├── questions.html
│   └── questionsDifficulty.html
├── ast_class.py
├── ast_class_tests.py
├── expressionCheckers.py
├── expressionCheckers_tests.py
├── expressionManipulations.py
├── expressionManipulations_tests.py
├── expressionPreProcessing.py
├── expressionPreProcessing_tests.py
├── flask_app.py
├── generators.py
├── generators_tests.py
├── logs.txt
├── README.txt
├── requirements.txt
├── stats.json
├── test_random.py
└── trees.py
```

Figure 1.6: Directory of the project

The project consists of 13 Python files, with 6 of them being test files and 7 containing the back-end functions of the tool.

There is a template sub-directory containing four HTML files using HTML and bootstrap. These files are for the front-end each representing the different pages: Home Screen, Analyse Expressions, Question Mode and Choosing the question difficulty.

1.5 Aim of the tool

The current aims focus on addressing the initial problem discussed as well ensuring the tool benefits as many students as possible.

Efficacy The tool should achieve the desired result of increasing the student's understanding of the key areas this tool focuses on: Boolean Expressions, Truth-Tables, converting expressions to CNF/DNF and Karnaugh maps.

Interactivity This is the main focus of an online tool in order to separate it from other learning mediums such as lecture recordings or readings. It is important the tool can interact with the user by being able to process a range of different user input and respond autonomously with useful feedback.

Usability A tool that has a good user interface will be adopted by more users. Very few users are likely to invest time in a complicated user interface since it could hinder their learning experience and the users may then prefer to switch to a more familiar traditional learning method.

Reliability The tool must produce the correct results consistently. This will help prevent misleading a student into believing a wrong result is correct or a correct result is wrong. This can affect user trust in the tool as well as impact the student's understanding of the content.

Chapter 2

Background

2.1 Overview of Part 1

2.1.1 Summary of work done in Part 1

The aim of the first part of the project was to design a teaching tool for any part of the Computation and Logic course. A significant amount of time was spent exploring different ideas and conducting user research to find which ideas work best for the students. Various parts of the course was explored, however from the user research, it was clear the idea to work on Boolean logic would be the most helpful for students.

The main focus after selecting Boolean logic was to get a very basic web tool hosted in time for students to test the tool before their exam and gain useful feedback through in-person discussion and surveys.

During the final stages of part 1 of the project, a basic web tool with the two main modes was constructed.

The first mode, being the Expression analyser, which could process a user expression or generate a random expression. The tool could then return either a corresponding truth table or steps of converting an expression to CNF.

The second mode, being the Answer Questions mode, which was limited to just asking students to convert a randomly generated expression to CNF.

2.1.2 Limitations of Part 1

There were many limitations of Part 1 that this project addresses.

The source code was all in one file which was not an issue during the initial build of part 1 due to the simplicity, however once the Answer Questions mode and the functions for CNF expression conversions were added, the readability suffered.

The user interface needed significant reworking. There was a lack of a navigation menu which makes it more complicated to navigate between the different modes. There was

a lack of a home screen resulting in users being directed to the analyse expressions mode by default.

In Answer Questions mode, all the question data such as the question, current difficulty and the question solution was all stored as URL parameters. This made URLs complicated and restricts all question data to be stored as strings only. This meant it was difficult to store tree structures which was integral to this project.

The random expression generator, which is a key part for making questions, was too random. This resulted in some very simple basic expressions and very complex expressions to be used in questions of the same difficulty. It was very clear a re-implementation of this generator was necessary.

The features in analyse expressions mode were limited, there is certainly scope to add more to this mode such as the use of Karnaugh maps and a converter for expressions to DNF.

The questions in Answer Questions mode had only one type of questions, which was asking students to convert to CNF. Not only was this very limited but also these questions are very time consuming to answer and it is unlikely a student would be very engaged.

The feedback generated on wrong answers in Answer Questions mode was also limited, which restricts the interactivity of the tool.

It was also found that the CNF converter had edge cases in which the minimal form of a CNF expression was not always returned. From further inspection of the code, it was found the current minimisation algorithm could not handle certain cases, such as $(A \vee B) \wedge (\neg A \vee B)$ could be simplified further to just B, but Part 1 of the tool did not return just B as the minimal form.

In the Informatics 1A course, they use XOR and Ternary operators in the course, which this tool was missing. It is important these operators are added as students have been asked to convert expressions that include these operators in previous exams, for example as seen in question 3, in the 'COMPUTATION AND LOGIC' December 2017 paper[2].

The variety of different syntaxes could be expanded further. For example the tool could not accept:

- `'/\'` or `'&&'` to represent the 'AND' operator
- `'\/'` or `'||'` to represent the 'OR' operator

2.1.3 Further Work discussed in Part 1

The main focus highlighted in Part 1 was for further User Testing, since as a result of the pandemic, user testing was difficult last year in March. While the pandemic is still problematic, there are now more opportunities for remote testing.

The other further work briefly outlined possible features that can be added. For example a much larger variety of different questions which has been highlighted as one

of the main limitations of Part 1, more detailed steps for CNF conversation and also looking into other forms such as DNF.

2.2 Existing Tools

In this section we will take a look at existing tools that involve interaction with Boolean Expressions.

2.2.1 Truth Table Generator - Stanford University

The Truth Table Generator by Stanford University [3] takes in an expression and produces a truth table as shown in Figure 2.1.

Truth Table Generator

This tool generates truth tables for propositional logic formulas. You can enter logical operators in several different formats. For example, the propositional formula $p \wedge q \rightarrow \neg r$ could be written as $p \wedge q \rightarrow \sim r$, as p and $q \Rightarrow$ not r , or as $p \ \&\& \ q \rightarrow \ !r$. The connectives \top and \perp can be entered as T and F.

a or b and c

a	b	c	$(a \vee (b \wedge c))$
F	F	F	F
F	F	T	F
F	T	F	F
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	T
T	T	T	T

Figure 2.1: Truth Table Generator by Stanford University

It consists of a basic and easy to use interface. This includes a text input box for entering an expression and a Truth Table that is generated as soon as input is detected.

The tool is simple, has a good user interface and is quite intuitive to use. There is a variety of expressions that can be entered and a variety of different syntax the user can use.

There are however limitations. Firstly, there are some alternative syntax that are not recognised by this tool. For example the tool cannot accept:

- '*', '.', and single '&' to represent the 'AND' operator
- '+' and single '|' to represent the 'OR' operator
- '-' to represent the 'Not' operator

While some of these syntax are rarely used, excluding them may impact usability for some students. It is also not clear what syntax is accepted by this tool without trial and error.

There are also some operators that are used in the course that this tool does not support, such as 'XOR' and 'Ternary Operators'. The tool in this project should include these operators to ensure students are prepared for dealing with these operators as well.

Overall this tool performs well, however is quite limited to the single functionality of displaying a Truth Table.

2.2.2 Truth Table Creator

The Truth Table Creator[4] takes in an Expression, generate a blank Truth Table and then asks the user to fill in the values for each row and also intermediate Truth Table values which represent sub-expression values. Some users would compute intermediate Truth Table values initially to help work out the overall Truth Table value. This tool then verifies if the user is correct.

It has two modes, one which gives intermediate feedback to the user as they are filling out the Truth Table and other where it waits till the User clicks 'Evaluate'. The interface is shown in Figure 2.2

The tool works very well, is interactive and is something that would help students learn more about Truth Tables.

There are a wide range of operators that tool accepts, this may however confuse students in Informatics 1A as they do not require to use all these operators and tool does also not use Ternary operators which is an operator that is used in Informatics 1A.

A major limitation is the very limited syntax accepted, it only accepts one particular syntax per operator. It also cannot accept spaces and only accepts atoms with one atom. These tough restrictions may make the tool harder to use for some students.

Overall this tool performs well and involves more interaction than just displaying a Truth Table, however this tool is still limited to only Truth Tables.

Truth Table Creator

Welcome to the interactive truth table app. This app is used for creating empty truth tables for you to fill out. Just enter a boolean expression below and it will break it apart into smaller subexpressions for you to solve in the truth table.

The app has two modes, immediate feedback and 'test' mode. Immediate feedback will immediately tell you when you get an answer wrong, while test mode won't tell you how many you got wrong until you submit the table. You can use the immediate feedback mode to practice, then use the test mode to make sure you understand everything.

To use the app, enter a boolean logic expression below. There is a legend to show you computer friendly ways to type each of the symbols that are normally used for boolean logic. Once you're done, pick which mode you want to use and create the table. Fill the tables with f's and t's and try to get all of the answers right. Good luck!

The source code for this project can be found here: <https://github.com/Thomas-Kim/truth-tables>

Do not insert spaces

Operator	What to Type
\vee	
\wedge	&
\rightarrow	->
\leftarrow	[-
\neg	!
\equiv	=
XOR	X
NAND	D
NOR	R

Expression: Want immediate feedback? ☒

Figure 2.2: Truth Table Creator

2.2.3 Logic calculator: Server-side Processing

This Logic calculator[5] takes in an expression and perform a large range of operations on the expression as shown in Figure 2.3.

The desired operation can be selected from the drop-down menu labelled: 'Task to be performed', the list of operations is shown in Figure 2.4

Logic calculator: Server-side Processing

[Help on syntax](#) - [Help on tasks](#) - [Other programs](#) - [Feedback](#) - [Deutsche Fassung](#)

▼ Examples and information on the input syntax

Please note that the letters "W" and "F" denote the constant values truth and falsehood and that the lower-case letter "v" denotes the disjunction. You may use all other letters of the English alphabet as propositional variables with upper-case letters being preferred.

Connectives must be entered as the strings "¬" or "~" (negation), "∧" or "&" (conjunction), "∨" or the lower-case letter "v" (disjunction), "→" or ">" (conditional), and "↔" or "<->" (biconditional). The English words "not", "and" and "or" will be accepted, too. Unicode characters "¬", "∧", "∨", "→" and "↔" require JavaScript to be enabled in your browser

P → ((Q → R) ∧ (¬S ∨ R))	(P → Q) ∨ (Q → P)	¬P → (P → Q)
Examples: (P → Q) ↔ (Q → P)	¬¬¬P → ¬((Q ∧ ¬R) ∨ (¬Q → R))	P → ¬Q
(A and B) or (C and not D)	(P1 and not P2) or (not P3 and not P4) or (P5 and P6) not (P and not P)	

For more details on syntax, refer to [Help on syntax](#).

enter a logical formula (see examples above)

¬ ∧ ∨ → ↔ () P Q R S T U V A B C D E G H

Task to be performed

Detailed truth table (showing intermediate results) ▼

Wait at most

30 seconds ▼

Execute

Reset

Figure 2.3: Logic calculator: Server-side Processing

Truth tables

Detailed truth table (showing intermediate results)

Truth table (final results only)

Transformations

Quine-McCluskey optimization

Atomic negations

Eliminate conditionals

Disjunctive normal form (DNF)

Canonical DNF (CDNF)

Conjunctive normal form (CNF)

Canonical CNF (CCNF)

Optimize expression (symbolically)

Optimize expression (symbolically and semantically - slow)

Notations

Polish notation

Graphical expression tree

Textual expression tree

Graphical alpha tree (Peirce)

Textual alpha tree (Peirce)

Graphical Begriffsschrift notation (Frege)

Properties/Checks

Tautology check

Prove the proposition

Figure 2.4: Logic calculator: Server-side Processing list of operations

It is clear this tool offers a very large range of functionalities. This can at the same time be overwhelming for some students especially if they are only interested in a few operations that are relevant for the course.

The accepted range of syntax is quite limited which can affect usability much like the Stanford tool. For example it cannot accept:

- '*' , '.' , '/' and '&&' to represent the 'AND' operator
- '\/' , '+' and '||' to represent the 'OR' operator
- '!' to represent the 'Not' operator

Also like the Stanford tool, this tool also does not accept 'XOR' and 'Ternary' operators.

While this tool does provide functionality to convert expressions to CNF and DNF, it does not show the steps to guide the students to the deduction.

Also the user interface could be improved, since the tool redirects the user to a new page to see the results of an operation, requiring the user to go back to access the original interface.

2.3 Existing Research

A limitation of part 1 of the project as previously discussed in section 2.1.2 was the previous minimisation algorithm not being able to handle particular edge cases.

There are alternative algorithms that could be implemented instead, as explored in the research paper: 'Optimization of the Quine-McCluskey Method for the Minimization of the Boolean Expressions'[6].

The goal of this paper was to explore algorithms that can minimise Boolean expressions and propose an optimisation for the Quine-McCluskey method.

The paper first introduces the Karnaugh Method, which involves constructing minimised CNF or DNF expressions from reading off the expression's Karnaugh Map.

This method is used in the Informatics 1A course, however there are limitations. It can be hard to use for expressions with more than four variables, as such expressions cannot be represented using a 2D Karnaugh Map. This will be problematic for this tool which is designed to accept an arbitrary number of atoms.

The alternative method: the Quine-McCluskey method, is a much more attractive option as it does not share the same limitation of the Karnaugh Method for the number of atoms that can be used.

The paper then proposes its own optimised Quine-McCluskey method and then compares the runtime to showcase the performance improvement.

From the paper it is clear that the Quine-McCluskey method would be good candidate to add to the minimisation step of the CNF/DNF converter.

Chapter 3

Refining the Logic Analyser

Part 2 focuses on addressing the aims of the tools, the limitations of Part 1 and also completing the further work addressed in Part 1

3.1 Summary of work done in Part 2

- Re-structured the project to improve readability and reduce total lines of code
- Re-design of the random expression generator for more consistency of the complexity of each generated expression
- Addition of new operators: 'XOR' and ternary operators
- Addition of new recognised syntax for 'AND' and 'OR'
- Redesigning the user interface: clearer instructions, navigation menu, separate home screen
- Redesign of user and question data storage
- Redesign of CNF minimisation function with final approach implementing the Quine-McCluskey method
- Improved autonomous feedback returned to users
- Addition of sub-questions and expression manipulation questions
- Addition of Truth-Table questions in Answer Questions mode
- Addition of Karnaugh maps to Answer Questions mode and Analyse Expressions mode
- Addition of DNF conversion to Answer Questions mode and Analyse Expressions mode
- Addition of randomly generated Unit Tests
- Addition of Unit Testing for white-box testing

- User testing from various students from Informatics 1A, Informatics 2D and Logic 1
- Created adapted version of the tool for Logic 1 course
- Think Aloud User Study

3.2 Project Restructure

The first stage in order to help redesign the project was to maximise readability of the current project from Part 1.

A significant amount of time was spent looking for duplicate code and finding ways to reduce the amount of code with the use of more functions.

The use of more functions helps clarify the higher level functionality of the code, especially with using functions with descriptive names.

The other key change was splitting the code up into multiple files. In where the previous main single python main file was reorganised into seven different files. This organisation made it more efficient to navigate to certain parts of code and reduced the amount of scrolling required.

3.3 Redesigning the Random expression generator

The basics of how the generator works, is that it randomly selects a Boolean operator and, by calling itself, it can generate sub-expressions. It will then return the overall expression by combining sub-expressions and a Boolean operator.

The base case would return a randomly selected atom, this is a random selection from ten characters: [P, Q, R, S, A, B, C, D, E, F]. These are all the atom names given in the December 2017 exam[2], therefore these atom names can be considered relevant to Informatics 1A students.

Selecting from ten possible characters would be a reasonable choice as well. If there too few choices then this will result in a higher occurrence of repeated atoms in an expression and if there too many choices, then expressions with repeated atoms would not be too common and students would receive less practise on such expressions which are still valid.

A significant flaw in the random expression generator in Part 1, was the inconsistent complexity of each expression generated. This is particularly problematic in the Answer Questions mode, since the expression should be consistent with the particular difficulty selected.

The exact solution to this problem was more challenging than previously thought and the random generator has been redesigned several times throughout the project.

The final solution implemented, involves a complexity weight which determines the complexity of the expression. For example, in this tool the weights for Very Easy,

Easy, Medium, Hard and Very Hard are 5, 10, 15, 20 and 25 respectively. The goal of the generator was to return an expression as random as possible whilst still achieving the desired complexity weight.

In this tool, the complexity weight of an expression is equal to the complexity weight of the operator plus the complexity weight of the sub-expressions. The weights assigned to each operator are shown in Figure 3.1.

Operator	Weighting
NOT	1
Single Atom	1
AND	2
OR	2
Single Implication	3
Ternary	7
XOR	8
Double Implication	8

Figure 3.1: Complexity weights of each of the Boolean Operators

The exact weights to choose for each operator can be subjective. However, after choosing the weights for 'NOT', 'AND' and 'OR', we can logically come up with weights for the other operators since all the other expressions can be expanded to an equivalent expression just using 'NOTs', 'ORs' and 'ANDs'.

For example the expression: $A ? B : C$ can be expanded to $A \wedge B \vee \neg A \wedge C$. We can see that the weighting from the operators add up to 7, therefore Ternary operators should also be weighted as 7.

Another issue to consider is how the weights of sub-expressions should be added. Looking at $A ? B : C$ again it can be seen when expanded, A is written twice. Therefore A should be given double the weight as students have to write that expression twice. So therefore the overall complexity weight of such an expression is $7 + 2*A + B + C$. Considering A, B and C have a weight of 1, then the minimum overall weight of a Ternary expression is 12. The minimum overall weight for other operators can be calculated similarly, depending on how such expressions would be expanded, Figure 3.2 shows the expanded forms chosen for this tool and the minimum overall weight of each expression.

Expression	Expanded Form	Minimum Overall Weighting
$\neg A$	$\neg A$	2
$A \wedge B$	$A \wedge B$	4
$A \vee B$	$A \vee B$	4
$A \rightarrow B$	$\neg A \vee B$	5
$A ? B : C$	$A \wedge B \vee \neg A \wedge C$	11
$A \oplus B$	$A \wedge \neg B \vee \neg A \wedge B$	12
$A \leftrightarrow B$	$(\neg A \vee B) \wedge (A \vee \neg B)$	12

Figure 3.2: Expansion of various expressions and minimum overall weighting

Overall, the function has to generate a random expression that matches the specified complexity weight. This can be achieved by randomly selecting an operator which has a minimum weight less than the specified weight. The function should then find sub-expressions such that the overall expression matches the specified complexity weight.

3.4 Additional syntax and operators

A limitation of part 1 of the project and existing tools is not recognising the 'XOR' symbol and the Ternary operators. This is understandably less common in Boolean logic in general.

However, the Informatics 1A course does feature these operators and so therefore their inclusion will benefit students more considering there is not a tool available that uses them.

The addition of 'XOR' was trivial whereas the addition to recognise Ternary operators was more complex due to the operator involving three sub-expressions which is unique to this operator only. This resulted in more fundamental changes in the tool's parser and various other functions.

The tool was limited in the possible syntax accepted. From exploring various tools it was clear more syntax could be recognised. The tool was updated to recognise the following additional syntax as shown in Figure 3.3

Operator	Added Syntax
AND	&& /\
XOR	\oplus XOR xor
OR	\/
Ternary Operator	? :

Figure 3.3: Additional Syntax Added in Part 2

Another change was converting all user input to Upper Case, this was to reduce any syntax errors related to case sensitivity. It is unlikely a user would use the same atom name twice with different cases but there is a good chance they will mistakenly disregard case for example when answering a question.

In the Informatics 1A course, upper case letters seem to be the most common names for atoms.

3.5 Redesigning the User Interface

One of the most important part of the interface is the instructions directing the User how to enter a Boolean expression in the tool. Without the use of instructions it is likely users could struggle with correct syntax, precedence, or may not be aware of all the operators they could use.

It is important these instructions are clear and the user is likely to read them. This can be achieved using a basic table that contains the minimum information such as the operators, the precedence and the accepted syntax.

A table was added replacing the less readable block of text. The table is visible to users on Analyse Expressions and Answer Questions mode. The table is shown in Figure 3.4

<u>Operator</u>	<u>Precedence</u>	<u>Accepted Syntax</u>
NOT	1	NOT not \neg - \sim !
AND	2	AND and \wedge . & && \wedge *
XOR	3	XOR xor \oplus
OR	4	OR or \vee + V
Implication	5	\rightarrow -> =>
Double Implication	6	\leftrightarrow <-> <=>
Ternary Operator	7	? :

Figure 3.4: Instructions shown to User

Another key change in layout was to introduce a 'Home Page'. As previously the home page was just the 'Analyse Expressions' page, which should be treated as one mode rather than the main mode.

The introduction of a 'Home Page' made it clear to users of the two separate modes of the tool.

The navigation menu was another key change as often modes can look alike due to similar layout and background. A navigation menu with a Tabs design helps display the current mode.

The navigation menu also makes it very easy to navigate to the desired mode.

3.6 Redesign of user and question data storage

For the Answer Questions mode to function, for each user, the current question needs to be stored, as well the solution to verify the question was correct and also the current difficulty the user is in for when the next question is generated.

The storage solution from Part 1 was to store all this information using URL parameters. However this was not an elegant solution as it required very large URLs that could only store String data type.

To simplify further work on the tool, the solution was to store the Question User data in a Python dictionary. This dictionary would have a randomly generated User ID as the key, and this key would be the only URL parameter. This resulted in a fixed length URL and the ability to store a significant amount of data using any type.

However to ensure that the server could handle the memory demands, the dictionary was limited to a size of 500 Users. This limit can easily be adjusted according the Server resources. An algorithm was implemented that would discard the least recently active User when the dictionary is over its limit. A limit of 500 was set because it is very unlikely there would be anywhere near 500 users in a day however user Data could accumulate of over time.

Now the tool can keep track of unique users, to understand how the tool is used, a 'logs.txt' file was produced. This would indicated how many unique users there are at particular time stamps, what expressions users are analysing, what questions they are given and what responses the questions get. This is all anonymous data, but the addition of the file proved to be useful to check the tool is properly used, check for common mistakes and also it can be used as reference for any bugs that are found.

Another useful file added was 'stats.json' file. This keeps track of the number of correct and incorrect answers for each difficulty. Earlier on in the project, it was not clear whether the questions were set at the correct difficulty and keeping track of this would help evaluate the tool during user testing.

3.7 Redesign of CNF Minimisation Function

As discussed previously, a limitation of Part 1 was the inability for the tool to handle some edge cases resulting in some non-minimal expressions being returned by the CNF minimisation function.

The first approach was to correct the function to handle such cases. This proved to be difficult as the current state of the function was very unnecessarily complex. A decision therefore was to redesign the function so that function is shorter, more readable and easier to modify.

The redesign took significant time and was a conceptional challenge. However, it became clear that handling the edge cases was complex and there it was difficult to check for other undiscovered edge cases as well.

Therefore after some research, the Quine McCluskey method was considered. This method was applied to converting expressions to a minimal DNF form. The method should work similarly for both forms but considering the literature was using minimal DNF forms, it was more clear how to implement.

Later converting a minimal DNF expression to a minimal CNF expression can be achieved using a more trivial procedure such as:

1. Find the Minimal DNF of the negated expression
2. Negate the result
3. Move the NOT inside the parenthesis
4. Remove any redundant NOTs

The DNF minimisation algorithm was implemented using the standard Quine McCluskey method. This process was complex and involved many steps and around 100 lines of code total.

Overall the method seems to work well and can solve the identified edge cases.

3.8 Improving User Interactivity

As one of the project aims is for increasing the interactivity, a key part of this is constructing more useful feedback to Users.

Previously in Part 1 of the project, when a user wanted to convert an expression to CNF, they would receive a standardised list of 5 steps in list.

Some of these steps were non-essential for a particular expression, for example step 1 was to remove double implications from an expression, and this step would display regardless of the expression, as shown in Figure 3.5 for the expression: $\neg a \rightarrow \neg d \wedge y$

1. Eliminate \leftrightarrow : $\neg a \rightarrow \neg d \wedge y$
2. Eliminate \rightarrow : $a \vee \neg d \wedge y$
3. Move \neg inwards : $a \vee \neg d \wedge y$
4. Distribute \vee over \wedge : $(a \vee \neg d) \wedge (a \vee y)$
5. Simplify CNF : $(a \vee \neg d) \wedge (a \vee y)$

Figure 3.5: The steps shown to Users in Part 1

Some of the steps were omitted, such removing redundant negations or showing all the distribution steps.

In Part 2 this feature was improved by using a dynamic table that would show only the relevant steps and also the extra steps not shown in the standard 5 steps in Part 1. This required some restructuring of the code, since previously the 5 steps were simply 5 variables mapped to 5 functions.

Now the tool loops through each stage, checks what is necessary to add to the feedback table. It then after each stage also checks if there are any redundant negations formed. Finally, rather than just adding the final distribution step, in more complex expressions the tool generates multiple distribution steps to clarify further to the User. On top of all this, there are also extra stages as a result of the new Ternary operators and XOR operators in the tool. Figure 3.6 shows the feedback for the same expression but with the Updated tool.

Step	Expression
	$\neg A \rightarrow \neg D \wedge Y$
Eliminate \rightarrow	$\neg\neg A \vee \neg D \wedge Y$
Remove redundant negation(s)	$A \vee \neg D \wedge Y$
Distribute \vee over \wedge	$(A \vee \neg D) \wedge (A \vee Y)$

Figure 3.6: The steps shown to Users in Part 2

The CNF steps have been updated in both the Analyse Expressions mode and the sample solutions for Answer Questions mode.

The other change in feedback, is for when a user answers a question wrongly. It was suggested that further feedback should be given to help the user figure out why they are wrong.

A new function for checking equality was created, which could check a two expressions are equal by performing an XOR operation between them. If the expressions are equal then the overall XOR operation should result in an expression which always evaluates to 0. If the expression evaluates to 1 then the function can return to the user when this is the case providing very useful feedback which proved to be very useful for participants trying out the tool.

3.9 Addition of Sub-Questions

Previously, questions in Answer Question mode were only asking students to convert an expression to CNF.

This not only makes the Answer Question mode limited but these questions were also cumbersome to answer, this is because converting an expression to CNF requires many steps.

A solution to both these problems is to split up the CNF conversion into sub-questions. This creates many different types of expression manipulation questions and these questions are also less cumbersome.

The solution involves:

1. Generating a random expression
2. Getting the stage of the random expression, where stage is the current step required to convert an expression to CNF
3. Get the user expression as input
4. If both user expression is equivalent and the user expression has progressed to a different stage, send a correct answer feedback
5. Repeat for the next sub-question until the user enters an expression in CNF form

3.10 Truth-Table Questions

A type of question that was added to Answer Questions mode was Truth-Table questions.

Its purpose was to help students understand truth-tables and also understand how to form expressions from them.

Truth table questions involve a randomly generated truth table and the student should enter any expression which satisfies it.

The user expression can then be used to create follow-up questions later on, if they are correct and are not in CNF or DNF.

The Truth-Table would have a set number of atoms: 2 for very easy and easy, 3 for medium, 4 for hard and very hard. The difficulty scales with the number of atoms used in the truth table.

When a Truth Table is generated, each Truth Table row is randomly assigned a result of 1 or 0. If by chance only 0s were assigned then a random row is selected and is assigned 1 similarly if only 1s were assigned then a random row is selected and is assigned 0.

A sample solution is generated which can be shown when requested by the user, this sample solution is also compared to the User expression to verify their answer.

To generate a sample solution, the tool loops through each row in the truth table, creating a conjunction of literals for each row. A disjunction of all the rows can be formed giving a solution.

However this solution although very simple to derive can be very long and cumbersome to check for students. Therefore, the tool would minimise this solution. It has a choice to return either DNF or CNF. It was decided the cleanest solution would be to perform both and check which solution has the fewest characters and therefore simpler for the student.

Whether a user receives a Truth Table question or an expression manipulation question is uniformly random, unless there is another sub-question generated from their previous answer, in which case they will be asked that.

3.11 The Addition of Karnaugh Maps

Karnaugh Maps was a new feature to both Analyse Expression mode and Answer Questions mode.

Karnaugh Maps are a key part in the course, it appears in many past papers and is also very compatible with the tool, therefore adding this feature made logical sense.

In Analyse Expressions mode, the Karnaugh Map can be generated corresponding to the entered expression.

The procedure to build a Karnaugh Map in this tool is as follows:

1. Get a list of the atoms used
2. First half of the atoms should be allocated for rows
3. Second half of the atoms should be allocated for columns
4. Calculate Row keys (Every Combination of row atoms values)
5. Calculate Column keys (Every Combination of column atoms values)
6. Fill the map by combining the row keys and column keys

The order of row keys and column keys have to be arranged in a very specific order such that students are able to see groupings. This was solved using a function to generate keys after a particular pattern was recognised.

Another consideration is what the tool should return should an expression be entered that has more than 4 atoms. Such a Karnaugh map cannot be represented in a 2D diagram in where a student is able to construct the most optimal groupings.

The decision was made to just expand the Karnaugh Map in a 2D direction, although this will not be helpful for students who wish to find minimal expressions.

An alternative would be to simply return an error, however an error would not help a student who is specifically looking to use more than four atoms in an expression.

Karnaugh Maps were also added to the Answer Question with a format very similar to Truth Table questions, where a random Karnaugh Map is generated and the student is tasked with finding an expression that satisfies it.

How the Karnaugh Map is generated is varied by difficulty as shown in Figure 3.7

Difficulty	Number of Atoms	Generation of Results
Very Easy	2	Uniformly Random*
Easy	3	Uniformly Random*
Medium	4	Random DNF of 2 conjunctions
Hard	4	Random DNF of 3 conjunctions
Very Hard	4	Uniformly Random*

Figure 3.7: Karnaugh Map by difficulty

*When all 0s generated randomly then a random result is flipped likewise for all 1s

To avoid confusion, the limit of number of atoms for a Karnaugh Map for Answer Questions mode is 4. Another method of varying difficulty is to vary the generation of the results.

The advantage is that the difficulty can vary significantly. With uniformly random distribution, it can very cumbersome in some cases to make groupings whereas by forcing a DNF of just 2 conjunctions, it is clear that there will only be 2 groupings students have to consider.

Like other questions, a sample solution had to be constructed to verify the Student answer and also give a solution should the student request for one.

For difficulties: Medium and Hard a sample solution already exists since a Karnaugh Map is made from a generated DNF. However for the other difficulties first a Karnaugh Map was generated and now a function is required to generate a corresponding expression.

To create an expression from a Karnaugh Map, the tool finds all the combinations of literals that evaluates to 1, these literals then form a conjunction. Once all the conjunctions have been found they can be linked together to form a DNF expression. Such an expression can put into the DNF minimisation function to return a minimal DNF expression to the student.

3.12 Addition of DNF conversion

Considering the tool has already functionality to minimise DNF expressions and students are almost as likely to work with DNF expressions, the addition of DNF functionality was a clear choice.

In Analyse Expressions mode a 'Convert to DNF' feature was added which works very similar to CNF.

The steps shown to a student are the same until the re-distribution of 'ANDs' and 'ORs'. The same function could be used, simply replacing where we see 'AND' in the code with 'OR' and similarly replacing where we see 'OR' in the code with 'AND'. Finally the last step was to return the minimised DNF expression which has already been implemented.

In Answer Questions mode, since CNF and DNF share many of the preliminary steps, the form is not considered until the expression reaches a stage in which case the tool randomly chooses between the two options and then asks the Student to convert an expression to the randomly chosen form.

In order to verify a Student's expression is in DNF, this requires the addition of a DNF checking function similar to the CNF checking function implemented in Part 1. Rather than starting a new function, this previous function was modified to check for both forms, and the particular form to check was a parameter to this function.

The advantage of modifying the CNF checker is that it reduces the need for unnecessary extra functions, however significant structural changes were required.

Chapter 4

Evaluation

4.1 Unit Testing

4.1.1 Unit Testing Framework

As the back-end of the project was developed using Python, there was a choice of possible unit testing framework libraries.

'unittest'[7] was used. This was a good choice as it was a part of the standard library in Python, it was easy to use, well documented and had all the relevant features required for this project.

4.1.2 Random Testing

The Random testing for this project resides in one file: 'test_random.py'.

This involves five different tests involving expressions with complexity weights matching the five difficulties from Very Easy to Very Hard.

Within each of the five tests, there are 160,000 sub-tests giving a total of 850,000 test cases in this test file.

The sub-tests are generated from 10,000 random expressions generated using the random expression generator for the particular complexity weight. For each random expression there were 16 tests performed, testing the particular expression manipulation functions that change the form of the expression but retain the equivalence of the original expression. 10 of the 16 tests were applying a particular manipulation and then asserting the modified expression remains equivalent. These include:

1. removing ternary operators
2. removing XOR operators
3. removing single implications
4. removing double implications

5. bringing negations inside brackets
6. removing redundant negations
7. redistribute to bring 'AND's outside the brackets
8. redistribute to bring 'OR's outside the brackets
9. convert to minimised CNF
10. convert to minimised DNF

6 tests were performed to check that the desired effect of some of these operations have been achieved, These include:

1. removing ternary operators
2. removing XOR operators
3. removing single implications
4. removing double implications
5. convert to minimised CNF
6. convert to minimised DNF

Only functions that were trivial to check were included, for example checking an expression contains a certain symbol or using the form checker function to check if an expression is in CNF or DNF.

All of the 850,000 test cases pass, which indicates reliability of the expression manipulation functions. These functions are core to this tool.

Overall random testing has proved to be successful as it is not possible to manually design 850,000 test cases in a reasonable amount of time. A large number of test cases is essential to show that the tool is very unlikely to produce an incorrect result.

There are limitations of random testing, such as:

- There are many functions that do not produce a result that can be verified autonomously. Manually assigning an expected result is not feasible for 850,000 test cases
- Even with a large number of test cases, random test cases can be ineffective at finding particular bugs that require a very specific input
- Random expressions generated may not reflect typical User input

4.1.3 White-box Testing

White-box testing involves designing test cases that considers the exact code. This will increase the chance of catching specific bugs that were previously not picked up by random testing. This is achieved by designing cases such that as many statements and branches that are reachable in the code are covered by the test cases.

For this stage of the project the test cases were manually constructed to ensure verifiable test cases for as many of the functions as possible.

For each Python file, a corresponding test file was created to cover all the functions in that file. The 'flask_app.py' was excluded as it required interaction with a live user, a browser, processing of HTML parameters and user UI making it more suitable for this part to be covered in User Testing. The 'trees.py' was excluded as it was used by other files and it was found to achieve 100% statement coverage from testing other files.

A test file has unit tests for all the functions excluding some that do not have a deterministic return values. Such functions involve the use of random functions, making it impossible to predict its result. Many statements that are not covered by white-box testing have been covered by user testing.

There are a total of 154 different white-box test cases split among 5 test files. All of them which pass, further indicating reliability of the code and no bugs present.

A standard method of evaluating how well unit tests test code, is by checking its statement coverage.

The Coverage.py[8] Python library was used to get the coverage statistics after each test file was run and the coverage for each file were:

- 'trees.py' has achieved 100% statement coverage from white-box testing as it was the most simple Python file and was used by many other functions being tested
- 'expressionCheckers.py', has achieved 100% statement coverage by its corresponding white-box testing file
- 'expressionManipulation.py', has achieved 100% statement coverage by its corresponding white-box testing file
- 'ast_class.py', has achieved 98% statement coverage by its corresponding white-box testing file. From analysis of the missed statements, it appears there is a section of code that was not reachable
- 'expressionPreProcessing.py', has achieved 97% statement coverage by its corresponding white-box testing file. From analysis of the missed statements, it appears some of the statements were not reachable
- 'generators.py' has only achieved a 22% statement coverage by its corresponding white-box testing file. This was a result of this file containing many functions with non-deterministic return values and therefore could not be verified correctly with unit testing

4.2 User Testing

4.2.1 Course Testing

Students from various courses were given access to the tool.

The intended target students to test the tool was Informatics 1A, however, their course only runs in Semester 1. For Semester 2, similar classes were considered as they would also benefit from using the tool to practise their courses.

The Informatics 1A students used the tool in December in preparation for their Exam. The version of the tool tested had the more up to date user interface, the inclusion of additional syntax and operators however there were no Karnargh Map or DNF features and the questions were limited to asking students to convert expressions to CNF. Their use of the Question Mode was observed through the 'stats.json' file with its content shown in Figure 4.1.

Question Difficulty	Number of correct answers	Number of incorrect answers
Very Easy	12	8
Easy	9	17
Medium	4	7
Hard	2	0
Very Hard	0	0

Figure 4.1: How Informatics 1A students answered questions

It appears that students had difficulty answering the questions as on 'Easy' and 'Medium' difficulty students were submitting more incorrect answers than correct answers. It is likely as a result of asking students to convert expressions to CNF which can be a lengthy process resulting in more room for error. This was the motivation to favour more questions that break down multi-step questions into sub-questions.

Informatics 2D students tested the tool during innovative learning week. Their version involved all the features as the final version with the exception of some refinements, such as the latest version of the random expression generator and other minor tweaks.

The decision to test Informatics 2D students, is that most of them would have previously studied Informatics 1A before and some parts are relevant for Informatics 2D as well. This tool aimed to then refresh some of relevant parts.

The tool was also given to Logic 1 students near the end of the teaching period. The main relevant part of the tool for those students were truth tables. To make sure the tool does not confuse the students, a modified version was hosted on <http://logic1test.pythonanywhere.com> which only includes Truth Table questions in Answer Questions mode and only Truth table and generate random expression operations in Analyse Expressions mode.

Overall course testing is an effective way to have the tool used by many students and will also benefit them.

There are however limitations, such as

- Students would use the tool but would not return feedback
- It is difficult to see how students interact with the tool exactly

4.2.2 Think Aloud Study

In this study, participants were selected to use the tool live and they were observed through screen share. They were also asked to share their thought process as they use the tool and they were asked questions throughout to explain how they feel about the features in the tool.

There were 8 participants selected who have a background in Computer Science but were not very familiar with Boolean logic. Therefore, these participants could learn or strengthen their understanding in the area.

All the questions and tasks given to each participant are included in Appendix A. No personal information, audio or video was recorded during the study to ensure the participant's anonymity. Each study lasted between 1hr to 1hr30.

4.2.2.1 Quantitative Analysis

To test efficacy of the tool, participants were asked to rate their understanding out of 10, in four relevant areas prior to and after using the tool. The average ratings for the 8 participants are shown in figure 4.2

	Mean Rating Before	Mean Rating After
Boolean Expressions	7.5	8.4
Converting expressions to DNF/CNF	4.6	6.5
Truth Tables	7.4	8.8
Karnaugh Maps	3	6.4

Figure 4.2: Participant average ratings, before and after using the tool

The ratings indicate the tool could be effective at improving participants' understanding in these key areas.

To keep a bias at a minimum, participants were not allowed to know what ratings they put before and the study was long enough which makes it difficult for the participant to remember their initial score, particularly as they were unaware that they will be asked the same question later on.

There is also the possibility of random variance in a participants response, however it is very unlikely that alone will increase the ratings across the different key areas for almost every participant.

Overall, from 8 participants, there is some indication from the quantitative data to suggest that this tool could help student understanding of Boolean Expressions, Converting expressions to DNF/CNF, Truth Tables and Karnaugh Maps.

4.2.2.2 Qualitative Analysis

Participants were asked to use the 'Analyse Expressions' mode and perform tasks to familiarise with all the features.

The first task involved construction of a truth table from their own expression, almost all participants were able to perform this task quickly and could produce a truth table they could recognise. Most participants responded saying it was clear but two of them were confused with the numbering of the rows on the left of the table.

Participants were then asked to generate a Karnaugh Map, this is something all participants could do. Participants mostly responded with the Karnaugh Map being clear or they could at least make sense of it if they were not too familiar with Karnaugh Maps.

Participants were then asked to convert expressions to CNF or DNF. Participants again were able to complete this task successfully. Some participants found the steps to convert an expression to CNF or DNF clear or helpful but there was more variety and some participants were confused. Suggestions included the use of colour, extra parenthesis or to show more steps.

Participants were then asked how they felt about the process of constructing their own expressions, all of them indicating it was clear or was easy.

The participants were then asked to use the Answer Questions mode and answer around three questions per difficulty. They were free to interact with the question how they like and they were observed and provided feedback after each question answered. They were asked at each difficulty how it matches their expectation.

The process of understanding the different questions and answering them seemed straightforward for most participants from observation and their responses.

Participants often struggled with some questions and the tool offered feedback for wrong answers in which some cases helped participants form a correct answer. There were cases when the participant had to check the sample solution generated by the tool, which they found helpful after attempting the question. Some participants suggested the use of hints rather than a sample solution.

Also some participants felt Truth-Table questions were more cumbersome and it was observed many preferred to skip these when the truth-table got too large.

When participants were asked to rate the difficulty of the questions to verify the difficulty was set correctly, there seems to be variability in difficulty but in most cases participants responded that they felt the difficulty matched their expectations.

Participants were given an opportunity to have free time with the tool to use it anyway they wish resulting in further general feedback. Participants felt there should be more explanations, examples and help features. Some pointed out repeated buttons in the navigation menu and the main interface. Almost all participants found the tool very simple to use.

Participants were asked to rate the usability, which most rated quite high. Participants would often comment on the simple and clean design. There were some comments on improvement such as addition of colours or icons and also there were comments on adding further clarity to the instructions.

Participants were asked to rate the interactivity of the tool, which many rated high. Some participants suggested more explanation or examples, should be given but most

felt the interactivity was sufficient and felt the feedback given in questions helped correct their mistakes.

Participants were asked what they learned, almost every participant was able to answer this and some learning a range of different things, demonstrating efficacy of the teaching tool.

Participants were finally asked how likely they were to use the tool had they have to study the course and every participant indicated they will use the tool.

Overall the 'Think Aloud' study took a substantial amount of time to set up, but it produced a very valuable insight on how users will use the tool. There were a lot of suggestions which would be useful for further work. The participants demonstrated the tool was effective, usable and interactive.

4.3 Project Completion

Relating back to the problem, this tool is a great candidate for Informatics 1A students to independently learn. From the user study it was clear how participants were able to grasp concepts about Computation and Logic quite well from using the tool.

Efficacy The tool as evident from the user study is able to increase the student's understanding of the key areas this tool focuses on: Boolean Expressions, Truth-Tables, converting expressions to CNF/DNF and Karnaugh maps.

Interactivity The user study has shown the tool is interactive with positive feedback given to the error correction and sample solution

Usability The tool was described as usable and very easy to use in the User Study.

Reliability The unit tests show the tool is able function very well

Bibliography

- [1] Michael Fourman. Informatics 1 - Introduction to Computation (2020-2021)[SEM1]. <https://course.inf.ed.ac.uk/inf1a>. [Online; accessed 08-January-2021].
- [2] INFR08012 INFORMATICS 1 - COMPUTATION AND LOGIC Tuesday 12 December 2017. https://www.inf.ed.ac.uk/teaching/exam_papers/2018/informatics1/solutions/inf1-cl_solutions.pdf. [Online; accessed 25-March-2021].
- [3] Truth Table Generator. <https://web.stanford.edu/class/cs103/tools/truth-table-tool/>. [Online; accessed 07-January-2021].
- [4] Thomas Kim. Truth Table Creator. <https://www.cs.utexas.edu/~learnlogic/truthtables/>. [Online; accessed 09-April-2021].
- [5] Logic calculator: Server-side Processing. <https://www.erpelstolz.at/gateway/formular-uk-zentral.html>. [Online; accessed 07-January-2021].
- [6] T. K. Jain, D. S. Kushwaha, and A. K. Misra. Optimization of the quine-mccluskey method for the minimization of the boolean expressions. In *Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08)*, pages 165–168, 2008.
- [7] unittest — Unit testing framework. <https://docs.python.org/3/library/unittest.html>. [Online; accessed 04-April-2021].
- [8] Coverage.py. <https://coverage.readthedocs.io/en/coverage-5.5/>. [Online; accessed 05-April-2021].

Appendices

Appendix A

Think Aloud study Questions/Tasks

Interview

How comfortable are you with Boolean Expressions?

[illegible]

How comfortable are you with converting expressions to CNF/DNF?

[illegible]

How comfortable are you with Truth Tables?

[illegible]

How comfortable are you with Karnaugh Maps?

[illegible]

Is it clear forming your own expressions? Can you generate one randomly now?

Your answer _____

Please answer 3 very easy questions in Question Mode, do you agree with the result, do you feel the difficulty matches your expectations?

Your answer _____

Please answer 3 easy questions in Question Mode, do you agree with the result, do you feel the difficulty matches your expectations?

Your answer _____

Please answer 3 Medium questions in Question Mode, do you agree with the result, do you feel the difficulty matches your expectations?

Your answer _____

Please answer 3 Hard questions in Question Mode, do you agree with the result, do you feel the difficulty matches your expectations?

Your answer _____

Please answer 3 Very Hard questions in Question Mode, do you agree with the result, do you feel the difficulty matches your expectations?

Your answer _____

On any question, look at a sample solution, is the solution useful?

Your answer _____

Please play around with the tool for how long as you would like, observe what they do and ask follow-up questions?

Your answer _____

Please rate the Usability of the tool? how usable was it, please point out features you liked or disliked, did you have any difficulties using the tool?

Your answer

Please rate the interactivity of the tool? Was the feedback useful, did you feel the tool was sufficiently interactive?

Your answer

How comfortable are you with Boolean Expressions

[illegible]

How comfortable are you with converting expressions to CNF/DNF?

1 2 3 4 5 6 7 8 9 10

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

How comfortable are you with Truth Tables?

1	2	3	4	5	6	7	8	9	10
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

How comfortable are you with Karnaugh Maps?

1	2	3	4	5	6	7	8	9	10
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

What have you learnt from using the tool and how likely are you to use it if you had to study Computation and Logic?

Your answer
